Linux embarqué

Avec deux études de cas

2e édition

Pierre Ficheux

EYROLLES

Linux
embarqué
2e édition

CHEZ LE MÊME ÉDITEUR

R. Hertzog, C. Le Bars, R. Mas. – **Debian** (collection Les cahiers de l'admin).
N°11639, 2005, 310 pages.

C. Blaess. – **Programmation en C sous Linux** – *Signaux, processus, threads IPC et sockets*.
N°11601, 2e édition, 2005, 964 pages.

P. Cegielski. – **Conception de système d'exploitation** – *Le cas Linux*.
N°11479, 2004, 680 pages.

J.-F. Bouchaudy, G. Goubet. – **Linux administration** (collection Guides de formation Tsoft).
N°11505, 2004, 936 pages.

R. Smith. – **Accroître les performances d'un système Linux** (collection Solutions Unix-Linux).
N°11430, 2004, 812 pages.

B. Boutherin, B. Delaunay. – **Sécuriser un réseau Linux** (collection Les cahiers de l'admin).
N°11445, 2004, 188 pages.

C. Blaess. – **Scripts sous Linux** – *Shell Bash, Sed, Awk, Perl, Tcl, Tk, Python, Ruby…*
N°11405, 2004, 762 pages.

Dans la collection Architecte logiciel

F. Vallée. – **UML pour les décideurs**.
N°11621, 2005, 282 pages.

J.-L. Bénard, L. Bossavit, R.Médina, D.Williams. – **Gestion de projet eXtreme Programming**.
N°11561, 2002, 300 pages.

Dans la collection Algorithmes

P. Naïm, P.-H. Wuillemin, P. Leray, O. Pourret, A.Becker. – **Réseaux bayésiens**.
N°11137, 2004, 224 pages.

G. Dreyfus, M. Samuelides, J.-M. Martinez, M. Gordon, F. Badran, S. Thiria, L. Hérault. – **Réseaux de neurones – *Méthodologies et applications***.
N°11464, 2004, 408 pages.

Y. Collette, P. Siarry. – **Optimisation multiobjectif**.
N°11168, 2002, 322 pages.

Dans la collection Accès libre

S. Gautier, C. Hardy, F. Labbe, M. Pinquier. – **OpenOffice. org 2.0 efficace.**
À paraître.

C. Gémy. – **Gimp 2 efficace**.
N°11666, 2005, 348 pages.

D. Garance, A.-L. Quatravaux, D. Quatravaux. – **Thunderbird** – *Le mail sûr et sans spam*.
N°11609, 2005, 300 pages.

T. Trubacz. – **Firefox** – *Retrouvez votre efficacité sur le Web !*
N°11604, 2005, 210 pages.

Linux embarqué

2e édition

Pierre Ficheux

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

© Groupe Eyrolles, 2002, 2005, ISBN : 2-212-11674-8

Remerciements

La réalisation d'un ouvrage, quel qu'en soit le sujet, est en soi un défi. Je voudrais ici remercier ceux et celles qui directement ou indirectement ont contribué à la finalisation de ce projet.

Je remercie tout d'abord Georges Noël, ancien professeur de mathématiques du lycée Michel-Montaigne, à Bordeaux, qui fut le premier à faire en sorte que l'ordinateur devienne un de mes fidèles compagnons de route. Joël Langla, professeur à l'École nationale d'arts et métiers de Bordeaux, est pour beaucoup dans la concrétisation de cette passion. À ces deux « maîtres » j'adresse mes plus respectueux sentiments.

Je remercie également ceux qui ont contribué à mon exploration du monde Linux et Open Source dans mon environnement professionnel : Daniel Roche, Éric Dumas, François Peiffer, Olivier Seston, Jean-Louis Heyd, Michel Stempin, Benoit Papillault, Éric Bénard, Romain Jean, Nicolas Saubade, Peter Williams. Je tiens à exprimer une pensée particulière pour René Cougnenc, pionnier de Linux en France et parti trop tôt.

Je remercie également le personnel de la société Open Wide pour son soutien durant ces mois de rédaction : Patrick Bénichou, François Hilaire, Wilfried Marek (merci pour « Brice de Nice »), Sylvie Plasse, Simon Boulay.

Je tiens à remercier Patrice Kadionik de l'ENSEIRB qui a su inspirer le chapitre sur µClinux, ainsi qu'Éric Bénard pour les exemples RedBoot.

Je remercie le personnel des Éditions Eyrolles, et notamment Muriel Shan Sei Fan pour sa patience et Éric Sulpice qui a initialisé le projet.

Je remercie la SNCF dont les navettes TGV ont favorisé la rédaction de cet ouvrage avec le concours d'IBM dont les batteries de ThinkPad durent au moins trois heures !

J'adresse aussi mes plus affectueux sentiments à Cathy, Maxime et Julie.

Enfin, merci à Raoul Volfoni et « Dirty » Harry Calahan.

Table des matières

TROISIÈME PARTIE

QUATRIÈME PARTIE

Préface

Nous sommes en 1995. Un industriel du sud-ouest de la France, leader sur un marché international, celui de la machine outil dédiée au monde de la confection, cherche à conforter le positionnement de ses solutions et à accroître son avance technologique. Il a compris qu'au-delà de la qualité et de la précision de ses machines, l'intégration au sein de ses produits des facteurs de compétitivité de ses clients serait un élément clé des compétitions à venir.

Ses systèmes, depuis la conception des modèles de vêtements, jusqu'à la découpe des pièces de tissus dont le tracé a été préalablement optimisé, intègrent une « intelligence » et une impressionnante sophistication pour le néophyte. Lorsque l'entreprise lance une consultation externe afin de se doter d'un outil logiciel de transformation et de conversion de multiples formats de fichiers graphiques, je ne me doute pas une seconde qu'un de ses responsables techniques, Pierre Ficheux, a entrepris de faire fonctionner ces différents équipements industriels sous LINUX. De mon côté, après une première analyse de l'objet de la consultation, une conviction s'impose : transformer des formats graphiques, somme toute répandus, est un travail qui a forcément déjà été réalisé. Des recherches menées en ce sens s'avèrent fructueuses et l'entreprise intégrera avant l'heure des briques de logiciels libres dans ses systèmes. On est loin à ce moment-là de la notoriété actuelle d'Internet, et encore plus de celle de l'Open Source. Il devient enfin naturel aujourd'hui de s'interroger et de rechercher ce qui a déjà été fait dans le domaine du logiciel, alors que ce dernier échappait jusqu'alors à une démarche pourtant banalisée dans le domaine scientifique ou artistique. Outre le fait qu'Internet et l'Open Source ont ouvert une voie de partage et d'échange sans précédent, des raisons plus profondes de ce retard sont peut-être à rechercher dans la nature même du logiciel et de son mode de production. Il reste qu'Internet et l'Open Source ont révolutionné le modèle centralisé de stockage et d'accès à l'information autour duquel était bâtie(s) tout aussi bien la bibliothèque d'Alexandrie ou les plus volumineuses de nos encyclopédies. L'accès à une information complètement distribuée, voire atomisée au niveau d'un individu a ouvert des champs infinis de partage et tend à faire émerger une véritable intelligence collective. La mondialisation ne peut pas avoir que des revers !

Les méthodes de travail dans de nombreux domaines en ont déjà été bouleversées et le mouvement est loin d'être stabilisé. Face à une énorme masse d'informations disponibles, les enjeux évoluent. Il s'agit maintenant de trier le grain de l'ivraie, de partager des

expériences analogues à ses propres préoccupations, c'est-à-dire ne plus partager simplement de l'information mais également de la connaissance. Alors que nous sommes au tout début de cette nouvelle histoire, et que les futurs réseaux sémantiques du net sont encore dans nos laboratoires, rédiger un ouvrage de synthèse sur un sujet pour le moins technique et pointu, reste une bonne façon de faire partager sa passion et son savoir.

Patrick Bénichou
PDG d'Open Wide
http://www.openwide.fr

Avant-propos

Cet ouvrage a pour but de proposer une méthodologie pour la création de systèmes embarqués avec Linux, en présentant notamment deux études de cas tirées du monde réel. De nombreux exemples de fichiers de configuration Linux, de codes source en C, langage de script Unix ou bien HTML agrémentent le tout. S'il faut choisir des qualificatifs pour cet ouvrage, les mots *concret* et *pragmatique* arrivent largement en tête !

L'ouvrage se veut aussi indépendant que possible des produits commerciaux, même si certains peuvent objecter ma préférence pour la distribution Red Hat Linux, citée plusieurs fois dans cet ouvrage. Il ne s'agit en rien d'une quelconque publicité déguisée ; Red Hat Linux étant, pour diverses raisons, fortement implanté dans le monde industriel, sa réussite commerciale a tendance à rassurer les industriels qui ne sont pas tous – loin s'en faut – des inconditionnels de Linux et de l'Open Source. Dans tous les cas, les concepts exprimés dans cet ouvrage sont valables quelle que soit la distribution utilisée, ce qui est logique puisque nous utilisons systématiquement le noyau Linux officiel.

À qui s'adresse ce livre ?

Ce livre s'adresse à un public qui désire se familiariser avec l'utilisation de Linux comme système embarqué, c'est-à-dire intégré à un équipement industriel dédié.

Il pourra intéresser dans sa première partie cadres et décideurs de départements techniques souhaitant évaluer l'état de l'art dans ce domaine ainsi que les produits commerciaux disponibles.

Une lecture plus poussée de l'ouvrage, dans ses parties suivantes, permettra aux développeurs de réaliser de façon pratique l'intégration d'un système Linux embarqué à partir de composants standards.

La lecture complète de l'ouvrage nécessite donc des notions de programmation en langage C et en langage de script Unix (Bourne Shell), ainsi que quelques connaissances générales en informatique industrielle. Cependant, un glossaire permettra au lecteur de se familiariser avec le vocabulaire spécifique bien souvent dérivé de la langue anglaise.

Les sources complets des exemples présentés sont disponibles en téléchargement sur le site d'accompagnement du livre à l'adresse *http://www.editions-eyrolles.com*.

Structure de l'ouvrage

L'ouvrage est divisé en quatre parties. La première partie traite des systèmes embarqués en général, de leur champ d'application ainsi que des avantages et inconvénients de l'utilisation de Linux pour ce type de système. Un chapitre décrit le matériel qui peut être utilisé pour un système Linux embarqué et un dernier chapitre donne quelques exemples de produits commerciaux utilisant Linux comme système d'exploitation embarqué. Cette première partie est relativement accessible d'un point de vue technique et ne demande pas de connaissances informatiques avancées.

La deuxième partie aborde la méthodologie de réalisation d'un système Linux embarqué à partir de composants standards comme le noyau Linux. Après une description de la structure de Linux, tant au niveau du noyau que de la répartition des fichiers système, les différentes phases de la création d'un système réduit sont abordées, en particulier la compréhension de la structure et du fonctionnement de Linux, l'optimisation de l'espace mémoire utilisé ou bien la création de scripts de démarrage. On s'attache à enrichir ce système minimal par une description détaillée des composants liés aux réseaux, à l'authentification des utilisateurs ou bien à l'installation du système sur des périphériques spéciaux comme les mémoires flash.

Dans une troisième partie, on détaille certaines mises en œuvre particulières, notamment pour les systèmes temps-réels, et l'on montre comment concevoir des interfaces graphiques lorsqu'elles sont nécessaires. Dans cette nouvelle édition, cette troisième partie contient également un chapitre consacré aux environnements de développement croisés permettant de produire et mettre au point des applications embarquées en utilisant Linux ou Windows comme système de développement.

Enfin, la quatrième partie est constituée de deux études de cas appliquant les concepts présentés dans la deuxième partie. Dans le droit fil de l'approche concrète qui est privilégiée dans cet ouvrage, on se propose de décrire la réalisation de la partie logicielle d'une platine de lecture/enregistrement de CD audio et fichiers au format MP3, puis d'un terminal de consultation Internet utilisant une version réduite du navigateur Netscape Communicator.

Précisions concernant cette deuxième édition

Depuis la sortie de la première édition de *Linux embarqué* (soit fin 2002), l'importance de Linux et des logiciels libres dans le monde industriel n'a cessée de croître. De nombreuses entreprises, y compris de grands noms de l'électronique grand public et professionnelle, ont adopté Linux comme système stratégique. Des actions significatives comme la création du CELF (C*onsumer Electronics Linux Forum*) montrent l'avancée inexorable de Linux dans ce domaine. Mieux encore, les principaux éditeurs des solutions embarquées propriétaires ont désormais tous une offre Linux, même s'ils continuent de maintenir et de promouvoir leur gamme classique. Enfin, la presse spécialisée s'est fait l'écho de cette évolution, ce qui m'a conduit à participer à de nombreuses conférences, articles et avis d'experts.

Cet ouvrage a, je l'espère, contribué à la promotion de Linux embarqué dans le monde francophone et j'ai reçu de nombreux courriers de soutien et de suggestion. Ce livre est devenu ce à quoi il était destiné, un support concret à la découverte des technologies Linux dans le monde industriel. J'ai dispensé depuis trois ans de nombreuses formations sur le sujet et ce livre constitue bien évidemment un support de cours directement utilisable.

En raison de la rapidité de l'évolution des techniques dans ce domaine, cette nouvelle édition était indispensable : elle est issue des remarques et questions collectées lors des nombreuses rencontres avec des utilisateurs. Concrètement, un nouveau chapitre est consacré à la création et à l'utilisation des chaînes de compilation croisée. De même, des points spécifiques concernant la prise en charge du noyau 2.6 sont présents dans les principaux chapitres de l'ouvrage. Le projet BusyBox, qui était en train de naître à l'époque de la publication de la première version de cet ouvrage, est devenu depuis lors une référence incontournable. Il est donc largement décrit dans cette nouvelle édition. Enfin, le chapitre décrivant le projet RTAI, qui est devenu depuis une référence dans le monde temps réel, a été largement mis à jour. Au niveau des exemples de réalisation, j'ai porté un regard particulier sur la Freebox développée par Free SA. Le célèbre terminal de connexion multimédia du premier opérateur Internet français est à ce jour un exemple d'innovation, d'autant que c'est un pur produit de l'Hexagone.

Je profite de cette nouvelle édition pour remercier ceux qui ont contribué de près ou de loin à la diffusion de cet ouvrage. Je remercie de nouveau l'équipe d'Open Wide dont l'intérêt pour Linux embarqué ne faiblit pas. Je remercie particulièrement Philippe Gerum pour son travail sur RTAI et sa maîtrise exceptionnelle des divers projets que nous avons eus à traiter. J'adresse également mes remerciements à Olivier Viné et Alexis Berlemont pour leur talent et leur dévouement.

Je remercie *Linux Magazine France* – et particulièrement Denis Bodor – qui m'offre régulièrement la possibilité de m'exprimer sur le sujet ainsi que Patrice Kadionik de l'ENSEIRB et Éric Bénard de la société EUKREA avec lesquels la collaboration est toujours fructueuse.

Enfin, je remercie Free SA et Maxime Bizon, pour les informations techniques relatives à la Freebox.

Pierre Ficheux
Août 2005

Introduction

Qu'est-ce que Linux ?

Linux est un système d'exploitation multitâche de la famille Unix. Il fut initialement développé sur processeur de type Intel x86 mais il a depuis été adapté sur un grand nombre d'architectures matérielles comme les PowerPC, SPARC, Alpha ou même des processeurs spécialisés et des micro-contrôleurs.

Linux est conforme à la norme Posix, ce qui signifie que les programmes développés sous Linux peuvent être recompilés facilement sur d'autres systèmes d'exploitation compatibles Posix.

Linux est également réputé pour sa grande interopérabilité, c'est-à-dire qu'il peut facilement s'intégrer dans un réseau informatique utilisant d'autres systèmes d'exploitation.

Le système d'exploitation Linux est libre, le code source des différents composants du système est disponible gratuitement sur le réseau Internet. Ce même code source peut être redistribué gratuitement en respectant les règles de la GPL (*General Public Licence*) et de sa variante, la LGPL (*Lesser General Public Licence*), toutes deux explicitées plus loin dans le présent chapitre ; elles sont définies par la FSF (*Free Software Foundation*) pour le projet GNU. Le principe général de la GPL est celui de la conservation de la liberté, chacun devant au moins redistribuer ce qu'il a reçu.

Qu'est-ce que l'Open Source[1] ?

Voici un rappel en neuf points de ce qui caractérise les produits Open Source.

1. *Libre redistribution*. La licence ne doit pas empêcher de vendre ou de donner le logiciel en tant que composant d'une distribution d'un ensemble contenant des programmes de diverses origines. La licence ne doit pas exiger que cette vente soit soumise à l'acquittement de droits d'auteur ou de royalties.

1. La définition du concept d'Open Source est disponible sur *http://www.opensource.org*. La traduction française est disponible sur *http://www.linux-france.org*.

2. *Inclusion du code source*. Le programme doit inclure le code source, et la distribution sous forme de code source, comme sous forme compilée, doit être autorisée. Quand une forme d'un produit n'est pas distribuée avec le code source correspondant, il doit exister un moyen clairement indiqué de télécharger le code source, depuis l'Internet, sans frais supplémentaires. Le code source est la forme la plus adéquate pour qu'un programmeur puisse modifier le programme. Il n'est pas autorisé de proposer un code source rendu difficile à comprendre. Il n'est pas autorisé de proposer des formes intermédiaires, comme ce qu'engendre un préprocesseur ou un traducteur automatique.

3. *Autorisation de travaux dérivés*. La licence doit autoriser les modifications et les travaux dérivés, et leur distribution sous les mêmes conditions que celles qu'autorise la licence du programme original.

4. *Intégrité du code source de l'auteur.* La licence ne peut restreindre la redistribution du code source sous forme modifiée que si elle autorise la distribution de fichiers *patches* au côté du code source dans le but de modifier le programme au moment de la construction. La licence doit explicitement permettre la distribution de logiciel construit à partir du code source modifié. La licence peut exiger que les travaux dérivés portent un nom différent ou un numéro de version distinct de ceux du logiciel original.

5. *Pas de discrimination entre les personnes ou les groupes*. La licence ne doit opérer aucune discrimination à l'encontre de personnes ou de groupes de personnes.

6. *Pas de discrimination entre les domaines d'application*. La licence ne doit pas limiter le champ d'application du programme. Par exemple, elle ne doit pas interdire l'utilisation du programme pour faire des affaires ou dans le cadre de la recherche génétique.

7. *Distribution systématique de la licence*. Les droits attachés au programme doivent s'appliquer à tous ceux à qui le programme est redistribué sans que ces parties ne doivent remplir les conditions d'une licence supplémentaire.

8. *La licence ne doit pas être spécifique à un produit*. Les droits attachés au programme ne doivent pas dépendre du fait que le programme fait partie d'une distribution logicielle spécifique. Si le programme est extrait de cette distribution et utilisé ou distribué selon les conditions de la licence du programme, toutes les parties auxquelles le programme est redistribué doivent bénéficier des droits accordés lorsque le programme est au sein de la distribution originale de logiciels.

9. *La licence ne doit pas contaminer d'autres logiciels*. La licence ne doit pas apposer de restrictions sur d'autres logiciels distribués avec le programme qu'elle couvre. Par exemple, la licence ne doit pas exiger que tous les programmes distribués grâce au même médium soient des logiciels « Open Source ».

Un peu d'histoire : de Minix à Linux

Le début d'une longue histoire…

Tout a commencé le 3 juillet 1991, dans le groupe de discussion *comp.os.minix*, Internet voit apparaître le message suivant :

```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: Gcc-1.40 and a posix-question
Message-ID: <1991Jul3.100050.9886@klaava.Helsinki.FI>
Date: 3 Jul 91 10:00:50 GMT

Hello netlanders,
Due to a project I'm working on (in minix), I'm interested in the posix
standard definition. Could somebody please point me to a (preferably)
machine-readable format of the latest posix rules? Ftp-sites would be nice.
```

Puis, le 25 août 1991 :

```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix -
I'm doing a (free) operating system (just a hobby, won't be big and
professional like gnu) for 386(486) AT clones. This has been brewing
since april, and is starting to get ready. I'd like any feedback on
things people like/dislike in minix, as my OS resembles it somewhat
(same physical layout of the file-system (due to practical reasons)
among other things).
I've currently ported bash(1.08) and gcc(1.40), and things seem to work.
This implies that I'll get something practical within a few months, and
I'd like to know what features most people would want. Any suggestions
are welcome, but I won't promise I'll implement them :-)
Linus (torvalds@kruuna.helsinki.fi)
PS. Yes - it's free of any minix code, and it has a multi-threaded fs.
It is NOT portable (uses 386 task switching etc), and it probably never
will support anything other than AT-harddisks, as that's all I have :-(.
```

En voici, plus ou moins fidèlement, une traduction dans la langue de Molière :

Bonjour aux utilisateurs du réseau,

Pour un projet sur lequel je travaille actuellement (MINIX), je suis intéressé par des informations concernant le standard Posix. Quelqu'un peut-il m'indiquer un pointeur vers une version électronique des dernières spécifications Posix ? Une adresse FTP serait le mieux.

Puis :

Bonjour à tous les utilisateurs de MINIX,

Je développe un système d'exploitation libre (juste pour le plaisir, ce n'est pas un gros projet comme GNU) pour les compatibles AT 386(486).

Cela infuse depuis le mois d'avril et ça commence à fonctionner. J'aimerais avoir votre avis concernant ce que vous aimez/n'aimez pas dans le système d'exploitation MINIX, vu que mon système lui ressemble (en particulier sur certaines couches physiques du système de fichiers et ce entre autres pour des raisons pratiques.

J'ai à ce jour porté bash(1.08) et gcc(1.40), et ça a l'air de fonctionner.

Cela signifie que j'aurai quelque chose d'utilisable dans quelques mois et j'aimerais savoir ce que désirent les utilisateurs. Toute suggestion est bienvenue mais je ne peux pas promettre de pouvoir la réaliser.

Linus (torvalds@kruuna.helsinki.fi)

P.-S. Oui, c'est indépendant de tout code MINIX, et cela utilise un système de fichiers multi-thread. Ce n'est PAS portable (utilisation du changement de contexte 386) et cela ne supportera vraisemblablement jamais autre chose que les disques durs AT, vu que je n'ai que ça.

Suite à cet article, la communauté des développeurs s'est intéressée de près au jeune étudiant finlandais dont le prénom était effectivement prédestiné à la réalisation d'un clone d'Unix. S'il s'était prénommé Robert ou Marcel, son travail n'aurait peut être pas connu le même succès…

Le système Minix que cite Linus Torvalds est également un système d'exploitation de la famille Unix développé par le professeur Andrew S. Tanenbaum de l'université d'Amsterdam (*ast@cs.vu.nl*) et destiné à des fins pédagogiques.

À l'époque de la parution de cet article, le monde Unix est emprisonné dans des solutions propriétaires, le plus souvent du fait des fabricants de matériel. Ces stations de travail ou serveurs sont très onéreuses, au moins dix fois le coût actuel du PC le plus puissant d'aujourd'hui. De même, le système d'exploitation diffusé avec ce matériel n'est jamais livré avec son code source car il est développé, le plus souvent, à partir de code source soumis à licence et provenant de la société AT&T (*American Telegraphs & Telephones*, *http://www.att.com*), créateur du système d'exploitation Unix.

Il existe cependant déjà à l'époque une version plus libre d'Unix, développée à l'université de Berkeley en Californie sous le nom d'*UNIX BSD* (*Berkeley Software Distribution*), et ce depuis la fin des années 1970. Ce produit majeur apportera à la communauté Unix des fonctionnalités essentielles comme la gestion du réseau TCP/IP. Cette version

est également à l'origine d'autres Unix libres ou produits commerciaux toujours utilisés aujourd'hui comme FreeBSD, NetBSD ou OpenBSD.

UNIX BSD est initialement adapté par le constructeur SUN Microsystems (*http://www.sun.com*) pour les premières versions de son système *SunOS*. Poussé par la pression commerciale, SUN décidera plus tard d'utiliser la version AT&T connue sous le nom de System V R4 et commercialisée par SUN sous l'appellation SOLARIS.

Le principal problème de l'époque est le rapport coût/performances du matériel disponible, ce qui contraint à l'utilisation de stations propriétaires pour des systèmes d'exploitation évolués tels qu'Unix. Les systèmes personnels tels que les IBM PC 386/486 sont de ce fait condamnés à utiliser des systèmes d'exploitation peu performants comme *MS-DOS* parfois associé à l'interface graphique MS-Windows *3*. Il en résulte que les clones d'Unix disponibles sur 386/486 comme MINIX ou bien MWC-COHERENT ne sont pas utilisables dans des environnements industriels.

Le fait est qu'un an et demi après l'apparition d'une première version 0.01 du noyau Linux qui tient sur un poster, le noyau 0.99 sort en décembre 1992. Ce dernier est déjà très complet, très stable, avec un support réseau TCP/IP de bonne qualité et une collection de pilotes de périphériques déjà non négligeable. Des distributions Linux comme la SLS ou la Slackware sont déjà disponibles. Ces dernières regroupent les composants Linux essentiels ainsi qu'une procédure d'installation facilitant la mise en place d'un système pour les non-spécialistes.

What is GNU ? Gnu is Not Unix

Il serait cependant faux de croire que le logiciel libre n'existait pas avant Linux. Déjà, dans les années 1980, Richard M. Stallman (*http://www.stallman.org*), un chercheur en intelligence artificielle du M.I.T (*Massachusetts Institute of Technology*), initialise un projet de développement de composants logiciels libres. De par son environnement culturel, Stallman s'oriente naturellement vers l'environnement Unix, non qu'il considère ce système comme parfait mais plutôt parce qu'il est selon ses dires *not so bad* (pas si mal). Le projet porte par ailleurs le nom de *GNU* (*http://www.gnu.org*), hommage au quadrupède cornu et velu du même nom (« gnou », en français), mais surtout définition récursive et typique de l'humour informaticien qui ne fait rire personne sauf les informaticiens eux-mêmes :

What is GNU ? Gnu is Not Unix.

Richard Stallman en profita pour mettre en place un nouveau type de licence de distribution appelée *GPL* (*General Public Licence*), utilisant le principe du *copyleft* par opposition au *copyright* des licences propriétaires. La description complète de la licence est assez longue et disponible en annexe de cet ouvrage. Il est cependant fortement recommandé de prendre connaissance de cette licence si vous désirez utiliser Linux ou un logiciel Open Source pour un projet industriel. Outre les éléments légaux indispensables, cette lecture vous apportera des éléments clés pour la compréhension de la philosophie du logiciel libre.

La GPL se différencie d'autres licences, plus permissives, permettant l'appropriation d'un code originellement libre mais pouvant ensuite être *capturé* par un éditeur de logiciels. La licence *LGPL* (*Lesser* GPL) est similaire à la GPL sur les points suivants :

- Le copyleft, qui interdit à quiconque de s'approprier le code source d'un logiciel et de ses dérivés modifiés à partir du moment où l'original a été diffusé sous GPL ou LGPL. Cela oblige en particulier à redistribuer le code source des modifications d'un composant déja placé sous GPL ou LGPL.
- La disponibilité des modifications : si vous effectuez une correction sur un programme et si vos corrections sont acceptées par le mainteneur, vous n'avez normalement donc plus à vous soucier de ces corrections pour les versions futures.

En revanche, la LGPL permet d'effectuer une édition de liens de code propriétaire avec des bibliothèques sous LGPL. Cette extension de la licence est fondamentale car elle permet la disponibilité sous Linux d'applications propriétaires qui utilisent des bibliothèques LGPL indispensables comme la glibc (GNU C library).

En raison des limitations matérielles citées précédemment, les composants du projet GNU sont utilisés sur des systèmes Unix propriétaires, ce qui permet déjà à l'époque de s'affranchir de certains outils payants mais pas forcément de meilleure qualité diffusés par les constructeurs. Le meilleur exemple en est le couple gcc/gdb (Gnu C Compiler/ Gnu DeBugger) considéré comme un des meilleurs compilateurs/débogueur sur le marché et surtout assurant une parfaite compatibilité entres les différentes plates-formes de développement.

Stallman lancera également le projet *GNU-Hurd*, focalisé sur le développement d'un système d'exploitation très novateur. Le projet est toujours actif actuellement mais n'a pas acquis une maturité industrielle comparable à celle de Linux ou d'autres clones libres d'Unix.

Il est important de noter que le travail de Linus Torvalds a d'emblée porté sur la partie noyau du système d'exploitation. Les autres utilitaires, comme gcc, gdb ou toutes les commandes Unix classiques, sont en fait empruntés au projet GNU ou à d'autres projets de logiciel libre comme l'interface graphique X Window System.

Le fait est qu'aujourd'hui Linux est connu par certaines ménagères, même de plus de cinquante ans et que GNU l'est beaucoup moins alors qu'un système tournant sous Linux contient beaucoup plus de code du projet GNU que du projet Linux lui-même.

On peut raisonnablement se demander comment un jeune étudiant finlandais de vingt-deux ans put ainsi damer le pion à un vieux routard du logiciel libre déjà célèbre comme l'était et l'est toujours Stallman. Les raisons sont multiples.

La première est chronologique : le projet Linux démarre au début des années 1990, à l'aube de l'explosion du réseau Internet et de l'économie dite nouvelle qui y est associée. La facilité de développement des projets communautaires comme Linux est augmentée de manière exponentielle grâce à ce nouveau moyen de communication. Ayant débuté son projet dix ans plus tôt, Richard Stallman n'a pu profiter des avancées d'Internet.

La deuxième raison tient à la personnalité des deux hommes. Stallman est une figure emblématique du logiciel libre mais que, peut être à raison, certains considèrent comme un extrémiste du libre. Son allure caricaturale de « gourou » exotique, ses positions plus que tranchées et ses coups de gueule spectaculaires ne pouvaient que susciter la méfiance du monde industriel.

Linus Torvalds est au contraire un garçon posé, calme et bien peigné, auquel les lunettes élégantes du futur gendre idéal siéent à ravir. Doté d'un sens inné de la communication et d'un charisme qui le positionnent naturellement dans un rôle de leader, il est aujourd'hui présenté par la presse comme celui qui révéla au monde une nouvelle conception du logiciel. Ce dernier point provoqua d'ailleurs la fureur de Stallman qui demanda instamment que le système dit Linux fût rebaptisé GNU-Linux.

La version 1.0 officielle sort en mars 1994 et Linux est d'ores et déjà utilisé pour des applications industrielles. La version 1.2 voit le jour en 1995. À ce moment, l'industrie informatique est de plus en plus sous l'emprise hégémonique des solutions Microsoft avec la sortie du fameux Window 95 et la consolidation de la version serveur du système de Microsoft Windows NT. Empêtré dans les architectures propriétaires très coûteuses et les guerres fratricides entre les différentes versions de système d'exploitation, le monde Unix est promis à une mort proche.

Pour l'instant, ce jeune Linux n'est pas encore, selon l'expression de Microsoft, *sur les écrans radar*, mais tout le monde est déjà convaincu que le salut d'Unix ne peut venir que de là. Les versions 2.0 et 2.2 de Linux sortent respectivement en juillet 1996 et janvier 1999. D'ores et déjà, de nombreuses distributions facilitant l'installation du système ont vu le jour, le support de périphériques sensibles comme les cartes graphiques est de bien meilleure qualité, et l'ont peut dire qu'à partir de 1999, les pilotes de périphériques sont fournis désormais par les fabricants de matériel, ce qui entérine l'adoption de Linux en tant que système d'exploitation majeur.

Les chiffres d'équipement des serveurs vendus durant l'année 1999 donnent 24,4 % à Linux, 37,8 % à Windows NT, 19,2 % à Novell Netware et seulement 15,2 % à tous les autres Unix.

Un autre élément déterminant des années 1999 et 2000 est l'adoption de Linux par les grands éditeurs professionnels de progiciels comme les systèmes de base de données, les logiciels de sauvegarde et autres composants essentiels de l'industrie informatique. Linux y perd la pureté d'un système informatique entièrement libre puisqu'il est désormais fréquent de faire cohabiter Linux avec des versions propriétaires de logiciels commerciaux.

Dans les cas les plus classiques d'utilisation de Linux, il est cependant possible de n'utiliser que des composants libres comme le serveur HTTP Apache, les langages de programmation Perl, PHP ou Python, ou bien les systèmes de base de données MySQL et PostgreSQL pour ne citer que les plus connus.

Outre la sortie du dernier noyau 2.4 en janvier 2001, les années 2000 et 2001 sont marquées par l'utilisation de plus en plus fréquente de Linux en tant que solutions industrielles

embarquées, c'est-à-dire optimisées pour l'exécution de tâches données. Même si la folie boursière associée aux jeunes sociétés proches de Linux est retombée comme un soufflé, ce dernier est aujourd'hui bien implanté dans le monde des serveurs.

IDC prévoit une croissance de Linux de 28,4 % dans ce domaine, contre 21,4 % pour Windows NT, et Linux se prépare une entrée fracassante à tous les niveaux d'un monde industriel où sa fiabilité, ses performances, sa facilité d'adaptation, sont autant d'avantages décisifs face à des solutions propriétaires.

Première partie

Systèmes embarqués, généralités

Cette partie s'attachera tout d'abord à décrire les concepts fondamentaux nécessaires à la compréhension de la problématique des systèmes embarqués. Nous décrirons ensuite les avantages et inconvénients du choix de Linux par rapport à ses principaux concurrents dans le domaine. Nous citerons de nombreux exemples de réalisations industrielles et embarquées à partir de Linux ainsi qu'un échantillon des solutions matérielles les mieux adaptées.

1

Les logiciels embarqués et leurs domaines d'application

Qu'est-ce qu'un logiciel embarqué ?

Un logiciel embarqué (*embedded software* en anglais) est un programme utilisé dans un équipement industriel ou un bien de consommation. La différence essentielle avec un logiciel classique tient à la complète intégration du logiciel embarqué dans cet équipement : il n'a pas de raison d'être en dehors de l'équipement pour lequel il a été conçu. On parle également de logiciel *intégré* ou *dédié*. Historiquement, cette notion est antérieure à l'idée même du logiciel tel qu'il est communément défini aujourd'hui : le programmateur mécanique du lave-linge ou de l'arrosage automatique fait partie des logiciels dédiés et personne n'achète un tel équipement pour son logiciel lui-même, mais bien évidemment pour la qualité des services que remplit l'équipement.

Cette logique a perduré et doit être prise en compte par les concepteurs des logiciels embarqués : l'équipement est valorisé uniquement par son aspect *fonctionnel* et un bon logiciel intégré le sera à un tel point qu'on finira par l'oublier ! Dans le cas de logiciels de très petite taille destinés à des tâches très spécifiques, on parle d'ailleurs en anglais de *deeply embedded software*, ce qui peut se traduire par *logiciel profondément enfoui*.

Quelles sont les caractéristiques d'un tel logiciel ?

Les points suivants permettent de caractériser un logiciel embarqué :

Ciblé

Son domaine d'action est *limité* aux fonctions pour lesquelles il a été créé. Concevoir un lave-linge qui fait le café n'est jamais une bonne idée…

Fiable et sécurisé

Le logiciel nécessite une grande fiabilité car il est destiné à un fonctionnement complètement autonome. Lorsqu'un piéton traverse la route, il n'est pas de bon ton que le logiciel de gestion de l'ABS (*Anti Blocking System*) de votre véhicule « s'excuse » de ne pouvoir fonctionner car *une erreur fatale a interrompu le fonctionnement du programme*. Cette contrainte en dit long sur le fossé qui existe par rapport aux logiciels classiques.

Maintenable dans le temps

Dans la majorité des domaines industriels, et en dehors du logiciel classique, la durée de vie des produits est longue de par des obligations légales ou du moins commerciales qui obligent l'industriel à maintenir le produit pendant une dizaine d'années, cas de l'industrie automobile. Dans le cas d'industries plus sensibles comme l'aéronautique, le militaire ou le spatial, cette durée peut être doublée. Il est donc indispensable que le logiciel embarqué soit maintenable durant toute la durée de vie du produit en cas de découverte de problème de fonctionnement majeur.

Spécifique

L'interface de dialogue avec l'utilisateur est spécifique : dans la majorité des cas, un tel logiciel n'utilise pas les interfaces classiques clavier/souris propres à la micro-informatique. S'ils existent, les périphériques d'affichage sont souvent limités à des panneaux de petite taille de type LCD (*Liquid Crystal Display*), et les périphériques d'entrée à quelques boutons poussoirs ou autres composants inhabituels dans l'informatique traditionnelle. Tout cela, combiné aux contraintes d'optimisation citées précédemment, nécessite une approche très *matérielle*, proche de l'électronique. Les concepteurs de logiciels embarqués sont rarement des informaticiens purs, plus souvent des êtres hybrides entachés de neurones d'électroniciens. De ce fait, ils sont habitués à des environnements de travail spartiates, pour ne pas dire arides, ce qui accentue encore le fossé avec le développeur standard habitué à un confort presque indécent.

Optimisé

Le plus souvent, mais ce n'est pas obligatoire, ce logiciel est de petite taille si on le compare aux volumes démesurés atteints par les logiciels classiques multi-usages comme en bureautique. La raison en est double :

- La nécessité de fiabilité citée précédemment cohabite mal avec un volume démesuré : plus on écrit de lignes de codes, plus on a de chances que celles-ci contiennent des *bogue*s.
- Le logiciel est souvent embarqué dans des équipements produits à grande échelle sur lesquels le moindre écart de coût dû à un embonpoint imprévu du logiciel peut avoir de fortes répercussions.

L'optimisation est également importante au niveau du temps de réponse. Le consommateur verra d'un très mauvais œil une soi-disant évolution technologique qui ralentit le fonctionnement de l'équipement. Outre l'exemple évident du freinage cité précédemment, un autre exemple marquant est celui des assistants personnels ou *PDA (Personal Digital Assistant)* : le logiciel Palm OS plus rustique que son rival Windows CE du pourtant hégémonique Microsoft lui tient la dragée haute, tout simplement parce que l'homme d'affaires pressé l'est suffisamment pour être agacé d'attendre le déroulement du menu *Démarrer.*

Alors que la micro-informatique a débuté avec 1 kilo-octet de mémoire vive et un système sur 8 Ko de ROM (le *ZX-81* !), il n'est pas rare aujourd'hui de voir des adolescents pester contre le PIII de l'année précédente qui n'affiche pas assez vite les formes plantureuses de Lara Croft.

Hélas, cette banalisation des performances a quelques effets pervers :

- Elle masque les imperfections et la faible optimisation (voire les *bogues*) de certains produits car comme dit le sage : « Software becomes slower faster than hardware becomes faster », ce qui peut se traduire par : « Le logiciel devient plus lent avant que le matériel ne devienne plus rapide », mais ça sonne mieux en anglais.
- Elle incite à une consommation effrénée de hardware, ce qui annule malheureusement la baisse des coûts de celui-ci.
- Elle donne des mauvaises habitudes au programmeur qui, fort de ses 512 Mo de RAM, 30 Go de disque et Pentium XXX à YYY GHz, ne comprend pas pourquoi la classe Java '*Hello World!*' prend tellement de temps à s'exécuter alors qu'il a suffi de cliquer sur le bouton *generate code*.
- Elle masque le fonctionnement réel du système, ce qui fait que l'on ne sait plus trop, des centaines de bibliothèques et des dizaines de packages, lesquels sont vraiment utiles pour l'utilisation courante de la machine. En revanche, en cas de vérification du disque après un crash système, on se rend compte que les packages sont bien là et que ces quatre cafés bus, en attendant, étaient très bons.

L'exemple le plus flagrant de cette course à la consommation est bien entendu les différentes moutures du système Microsoft Windows et les applications associées pour lesquelles chaque version est systématiquement accompagnée d'une augmentation de taille, compensée bien sûr par l'achat de quelques giga-octets de disque dur ou d'une barrette de mémoire supplémentaire.

Logiciel embarqué ou système embarqué ?

Nous avons pour l'instant parlé de *logiciel* embarqué alors qu'il est fréquent d'entendre la terminologie de *système* embarqué. Cette terminologie désigne le plus souvent un *système d'exploitation*, version complexe et multi-usage du concept de logiciel. Si nous revenons à la notion de système d'exploitation telle qu'elle est communément admise, référence est alors faite à un ensemble de programmes permettant :

1. de gérer les ressources de l'installation matérielle en assurant leurs partages entre un ensemble plus ou moins grand d'utilisateurs ;
2. d'assurer un ensemble de services en présentant aux utilisateurs une interface mieux adaptée à leurs besoins que celle de la machine physique.

La première réaction, légitime, est de considérer qu'un système d'exploitation est *a priori* beaucoup trop complexe et surdimensionné pour remplir les tâches décrites dans la section précédente. C'est vrai dans certains cas de spécificité extrême du logiciel à embarquer ou de fortes contraintes matérielles mais il est également vrai que l'amélioration des performances du matériel permet dans un grand nombre de cas d'utiliser un système d'exploitation adapté au lieu d'un simple logiciel dédié.

Les avantages de l'utilisation d'un système d'exploitation sont les suivants :

- Comme dans le cas du développement de logiciel classique, il affranchit le développeur de l'applicatif embarqué d'un travail d'adaptation très proche du matériel, ce qui permet de diminuer le temps de développement et donc les coûts. L'écriture du support de standards du marché comme les bus PCI ou USB est extrêmement lourde en cas de non-utilisation d'un système d'exploitation. Dans le cas d'un système, ce travail est réalisé par une autre équipe spécialisée ou un fournisseur externe.
- Si le système d'exploitation utilisé est suffisamment répandu, il permet aux applications industrielles et embarquées de bénéficier des mêmes avancées technologiques que les applications classiques. C'est ainsi qu'il est aujourd'hui possible d'utiliser dans des systèmes réduits des protocoles de communication hérités de l'informatique classique et du multimédia. Nous pouvons citer par exemple l'utilisation généralisée du protocole TCP/IP et de ses dérivés comme HTTP (*Hyper Text Transfer Protocol*) ou FTP (*File Tranfer Protocol*) dans des procédures de communication entre des systèmes classiques et des micro-systèmes dédiés. La complexité des protocoles de communication et donc le temps de mise au point d'une version spécifiquement adaptée pour un logiciel embarqué rendent ce choix techniquement et économiquement très hasardeux. L'utilisation d'un système d'exploitation qui inclut un support natif et largement débogué de ces protocoles est alors un bien meilleur choix car le support d'un protocole se réduira le plus souvent à l'ajout d'un module ou d'un programme externe déjà testé, comparé aux nombreuses heures de mise au point nécessaires à la mise en place d'une version « maison ».

Remarque

HTTP est le protocole utilisé pour le transfert des données multimédias entre un serveur web et un poste client équipé d'un navigateur ou *browser.* Plus ancien, le protocole FTP est lui réservé au simple transfert de fichier. L'utilisation dans l'embarqué de ces protocoles standards en remplacement de protocoles propriétaires facilite l'interopérabilité des systèmes.

- Les systèmes d'exploitation fournissent en général un environnement de développement facilitant la mise au point des programmes dans un contexte beaucoup plus accueillant et performant que le système cible – celui sur lequel est censé tourner le

programme définitif. Un exemple industriel célèbre est celui de la PlayStation2 de SONY. Les développeurs de jeux ne pouvant disposer de la console définitive immédiatement, ils utilisèrent un environnement de développement basé sur Red Hat Linux permettant de simuler le fonctionnement de la console sur des stations de travail.

Remarque

Une étude de cas, dans la troisième partie de l'ouvrage, décrira également une méthode simple utilisée pour développer une application embarquée contrôlant un écran *LCD* sans pour cela disposer de ce matériel.

A contrario, le principal inconvénient de l'utilisation d'un véritable système d'exploitation proche d'un système classique est la taille mémoire nécessaire. Il est bien évident que, si l'espace disponible est réduit à quelques centaines d'octets pour accomplir une tâche rudimentaire, un vrai système d'exploitation ne s'imposera pas.

Les champs d'application

Comme nous l'avons précisé au début de ce chapitre, le champ d'application des systèmes embarqués est très vaste. Le fait est qu'il est d'ailleurs de plus en plus vaste car de nombreuses fonctions autrefois réalisées par des systèmes mécaniques ou du moins *analogiques* sont aujourd'hui remplacées par des composants logiciels, même rudimentaires.

Au niveau applicatif, les systèmes embarqués se retrouvent historiquement sur les sujets suivants (liste non exhaustive) :

- contrôle de processus industriels ;
- commande numérique, machines outils ;
- automobile ;
- télécommunications : centraux téléphoniques, téléphones mobiles ;
- réseaux informatiques : routeurs, systèmes ;
- périphériques informatiques : imprimantes, photocopieurs ;
- aéronautique et transports en général ;
- systèmes médicaux.

À titre d'exemple, on peut citer des produits comme :

- l'autoradio MP3 de chez EMPEG (*http://www.empeg.com*) ;
- les magnétoscopes numériques comme TiVo (*http://www.tivo.com*) ou ReplayTV (*http://www.replaytv.com*). (Ces produits seront un peu plus détaillés en fin de première partie de l'ouvrage) ;
- les platines CD/MP3, dont un prototype de développement sera présenté comme étude de cas dans la troisième partie de cet ouvrage.

D'un point de vue technique, le champ d'application peut être grossièrement divisé en deux grandes familles :

- le contrôle de processus sans contrainte ou à faible contrainte temps réel ;
- le contrôle de processus avec contrainte temps réel.

Cette séparation est fondamentale car elle déterminera complètement le choix des composants logiciels à utiliser. Dans le premier cas, on pourra envisager l'utilisation d'un système d'exploitation dérivé d'un système classique comme Linux. L'adaptation se situera principalement au niveau du développement de pilotes de périphériques et de l'optimisation du système en taille ou en performances. Dans le second cas, les contraintes matérielles nécessiteront l'utilisation de composants spécialisés, soit un logiciel spécifique, soit un système d'exploitation dit *temps réel* (*Real Time Operating System* ou RTOS). Les définitions et comparaisons des deux types de systèmes seront approfondies dans la section suivante.

Le domaine de l'équipement grand public avait échappé aux systèmes embarqués d'abord pour des raisons de coût du matériel. De plus, ces équipements fonctionnaient de manière isolée et n'avaient jusqu'ici aucun lien avec les réseaux informatiques. Le développement des services sur Internet a incité les industriels à intégrer des produits initialement peu communicants dans des environnements en réseau. Cette intégration nécessite l'utilisation de protocoles de communication hérités de l'informatique, et donc le plus souvent d'intégrer des couches logicielles supportant ces protocoles.

On peut même citer des produits aussi peu excitants que les réfrigérateurs dont les modèles futurs pourront intégrer des fonctionnalités réseau, un navigateur Internet et donc un système d'exploitation évolué. À titre d'exemple, on peut citer le ScreenFridge de chez Electrolux (*http://www.electrolux.co.uk/screenfridge*).

L'émergence de ces produits est également liée à l'apparition d'un concept né de l'économie du réseau : l'*appliance*. Cette notion est difficilement traduisible directement en français mais afin d'éclairer le lecteur nous présentons ci-après un appliance spécialisé dans la fabrication du café :

Figure 1-1
Appliance de fabrication du café

Qu'est-ce qu'un *appliance* ?

Le site de la version française du *Jargon informatique* (*http ://www.linux-france.org/prj/jargonf*) en donne la définition suivante : « Se dit de toutes sortes de machines dont la principale caractéristique est de pouvoir être (théoriquement) simplement branchées pour fonctionner immédiatement de manière parfaitement opérationnelle. Le plus fou, c'est que cette promesse est souvent tenue ! *Serveur applicatif* en français ».

La notion de *serveur applicatif* découle d'un besoin très simple du marché informatique qui est celui de considérer les services logiciels (serveur web, serveur FTP, base de données) comme des éléments modulaires et facilement administrables le plus souvent grâce à l'outil universel qu'est le navigateur Internet. Si le besoin de service se fait sentir, nul besoin d'installer un système d'exploitation compliqué sur lequel il faudra installer un logiciel qui l'est encore plus. Dans un appliance, l'administrateur ne saura pas forcément quel type de système d'exploitation est utilisé dans la « boîte noire ». Il suffit d'ajouter un élément, souvent sous forme de *rack* et de choisir une adresse IP sur le réseau local à moins que l'élément en question ne sache la déterminer tout seul grâce à un serveur DHCP ou BOOTP. Tout cela est effectué grâce à des afficheurs LCD en face avant ou un navigateur Internet exécuté sur une machine tierce dont le système d'exploitation importe également peu.

Rappel

Les protocoles DHCP (*Dynamic Host Configuration Protocol*) et BOOTP (*BOOTstrap Protocol*) permettent à des systèmes connectés à un réseau de type TCP/IP de récupérer automatiquement une adresse IP sans intervention d'un opérateur ni configuration. Cette adresse leur sera affectée par un *serveur* DHCP ou BOOTP.

On voit donc que la notion d'appliance est fortement liée à celle d'un système embarqué de par les contraintes de facilité d'installation et d'utilisation, de transparence et de sûreté de fonctionnement. De ce fait, la notion d'appliance dépasse les limites du monde informatique. En effet, si le système *home cinema* du futur inclut un navigateur web ou même un serveur web utilisable par les autres postes multimédias du réseau local, il y aura de moins en moins de différence entre ce serveur familial et les millions d'autres présents sur Internet. Les logiciels utilisés seront souvent les mêmes et le système d'exploitation sera également le même : Linux !

Remarque

Nous rejoignons ici le concept des *milliards de nœuds* d'Internet prédit par le chercheur français Christian Huitema dans son ouvrage de vulgarisation *Et Dieu créa l'Internet* (Éditions Eyrolles, 1995) : « Il y a déjà des microprocesseurs, en fait de tout petits ordinateurs dans bien d'autres endroits [...]. D'ici quelques années, le développement et les progrès de l'électronique aidant, ces microprocesseurs deviendront sans doute de vrais ordinateurs élaborés et il sera tout à fait raisonnable de les connecter à Internet » [HUITEMA 95].

Typologie des systèmes embarqués

Comme nous l'avons rapidement signalé dans la section précédente, les systèmes embarqués se divisent en deux familles : les systèmes dits *temps réel* et les autres. Nous allons expliquer dans cette section les différents modes de partage du temps dans ces deux types de systèmes d'exploitation.

Temps partagé et temps réel

La gestion du temps est un des problèmes majeurs des systèmes d'exploitation. La raison en est simple : les systèmes d'exploitation modernes sont tous *multitâches*, or ils utilisent du matériel basé sur des processeurs qui ne le sont pas, ce qui oblige le système à partager le temps du processeur entre les différentes tâches. Cette notion de partage implique une gestion du passage d'une tâche à l'autre qui est effectuée par un ensemble d'algorithmes appelé *ordonnanceur* (ou *scheduler*).

Un système d'exploitation classique comme Unix, Linux ou Windows utilise la notion de *temps partagé*, par opposition *au temps réel*. Dans ce type de système, le but de l'ordonnanceur est de donner à l'utilisateur une impression de confort d'utilisation tout en assurant que toutes les tâches demandées sont finalement exécutées. Ce type d'approche entraîne une grande complexité dans la structure même de l'ordonnanceur qui doit tenir compte de notions comme la régulation de la charge du système ou la date depuis laquelle une tâche donnée est en cours d'exécution. De ce fait, on peut noter plusieurs limitations par rapport à la gestion du temps.

Tout d'abord, la notion de priorité entre les tâches est peu prise en compte, car l'ordonnanceur a pour but premier le partage équitable du temps entre les différentes tâches du système (on parle de *quantum* de temps). Notez que sur les différentes versions d'Unix dont Linux, la commande `nice` permet de modifier la priorité de la tâche lors du lancement.

Ensuite, les différentes tâches doivent accéder à des ressources dites *partagées*, ce qui entraîne des incertitudes temporelles. Si une des tâches effectue une écriture sur le disque dur, ce dernier n'est plus disponible pour les autres tâches à un instant donné et le délai de disponibilité du périphérique n'est pas prévisible.

> **Rappel**
> La gestion des ressources partagées entre différentes tâches se fera grâce à des composants appelés *sémaphores*.

En outre, la gestion des entrées/sorties peut générer des temps morts, car une tâche peut être bloquée en attente d'accès à un élément d'entrée/sortie.

La gestion des *interruptions* reçues par une tâche n'est pas optimisée. Le temps de latence – soit le temps écoulé entre la réception de l'interruption et son traitement – n'est

pas garanti par le système. Par comparaison, le temps de latence dans le cas d'un système temps réel est souvent inférieur à 100 microsecondes.

Enfin, l'utilisation du mécanisme de *mémoire virtuelle* peut entraîner des fluctuations dans les temps d'exécution des tâches.

Rappel

La mémoire virtuelle permet au système de disposer d'une quantité de mémoire, supérieure à la quantité de mémoire vive réellement disponible. Le système pourra pour cela utiliser des espaces dédiés sur la mémoire de masse afin de simuler une mémoire *virtuelle*. L'inconvénient en sera bien sûr une forte dégradation des performances.

Toutes ces limitations font que le temps de réponse d'un système classique n'est pas garanti.

Remarque

Une grande partie des systèmes industriels ne nécessitent pas forcément une gestion stricte du temps, ce qui permettra d'utiliser les systèmes classiques avec peu de modifications. Si la tâche du système est bien définie, on pourra en particulier connaître à l'avance l'espace mémoire utilisé et s'affranchir de l'utilisation de mémoire virtuelle.

Le cas des systèmes temps réel est différent. Il existe un grand nombre de définitions d'un système dit *temps réel,* et la plupart d'entre elles sont contradictoires. Une définition simple d'un tel système pourra être la suivante :

« Un système est dit temps réel lorsqu'il est soumis à des contraintes de temps et qu'il y répond dans un intervalle acceptable » [BLACHIER 2000],

ou bien :

« Un système temps réel est une association logiciel/matériel où le logiciel permet, entre autres, une gestion adéquate des ressources matérielles en vue de remplir certaines tâches ou fonctions dans des limites temporelles bien précises » [MABILLEAU 2001].

Il est évident que la structure de ce système dépendra de ces fameuses contraintes. On pourra diviser les systèmes en deux catégories :

- Les systèmes dits à contraintes *souples* ou *molles* (*soft real time*). Ces systèmes acceptent des variations dans le traitement des données de l'ordre de la demi-seconde (ou 500 ms) ou la seconde. On peut citer l'exemple des systèmes multimédias : si quelques images ne sont pas affichées, cela ne met pas en péril le fonctionnement correct de l'ensemble du système. Ces systèmes se rapprochent fortement des systèmes d'exploitation classiques à temps partagé.
- Les systèmes dits à contraintes *dures* (*hard real time*) pour lesquels une gestion stricte du temps est nécessaire pour conserver l'intégrité du service rendu. On citera en guise

d'exemples les contrôles de processus industriels sensibles comme la régulation des centrales nucléaires ou les systèmes embarqués utilisés dans l'aéronautique.

Les systèmes à contraintes dures doivent répondre à trois critères fondamentaux :

1. Le déterminisme *logique* : les mêmes entrées appliquées au système doivent produire les mêmes effets.
2. Le déterminisme *temporel* : une tâche donnée doit obligatoirement être exécutée dans les délais impartis ; on parle d'*échéance*.
3. La *fiabilité* : le système doit être disponible. Cette contrainte est très forte dans le cas d'un environnement embarqué car les interventions d'un opérateur sont très difficiles ou même impossibles. Cette contrainte est indépendante de la notion de temps réel mais la fiabilité du système sera d'autant plus mise à l'épreuve dans le cas de contraintes dures.

En résumé, on peut dire qu'un système temps réel doit être prévisible (*predictible*), les contraintes temporelles pouvant s'échelonner entre quelques microsecondes et quelques secondes.

Attention

Un système temps réel n'est pas forcément plus rapide qu'un système à temps partagé. Il devra en revanche satisfaire à des contraintes temporelles prévues à l'avance.

La petite expérience décrite ci-après met en évidence la différence entre un système classique et un système temps réel. Considérons la configuration décrite sur la figure 1-2 [BLACHIER 2000] :

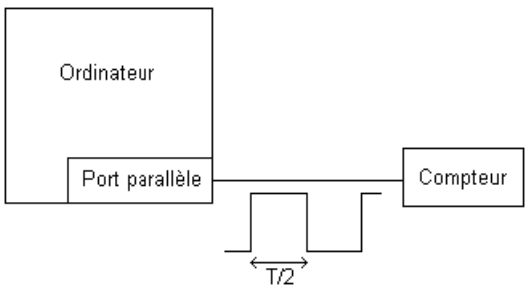


Figure 1-2

Test comparatif temps réel/temps partagé

Le but de l'expérience est de générer un signal périodique sortant du port parallèle du PC. Le temps qui sépare deux émissions du signal sera mesuré à l'aide d'un compteur. Le but est de visualiser l'évolution de ce délai en fonction de la charge du système. La fréquence initiale du signal est de 25 Hz (Hertz), ce qui donne une demi-période T/2 de 20 ms (milli-secondes).

Sur un système classique, cette demi-période varie de 17 à 23 ms, ce qui donne une variation de fréquence entre 22 Hz et 29 Hz. La figure 1-3 ci-après donne la représentation graphique de la mesure sur un système non chargé, puis à pleine charge :

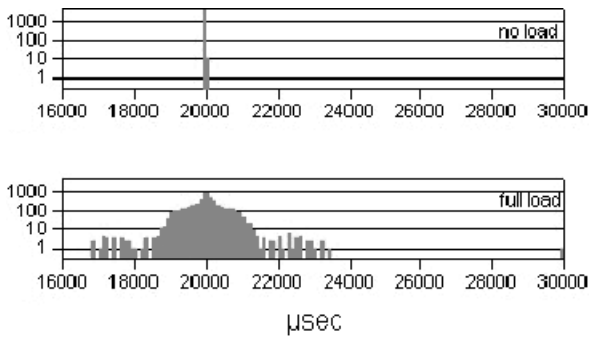


Figure 1-3

Représentation sur un système classique

Sur un système temps réel, la demi-période varie entre 19,990 ms et 20,015 ms, ce qui donne une variation de fréquence de 24,98 Hz à 25,01 Hz. La variation est donc beaucoup plus faible. La figure 1-4 donne la représentation graphique de la mesure :

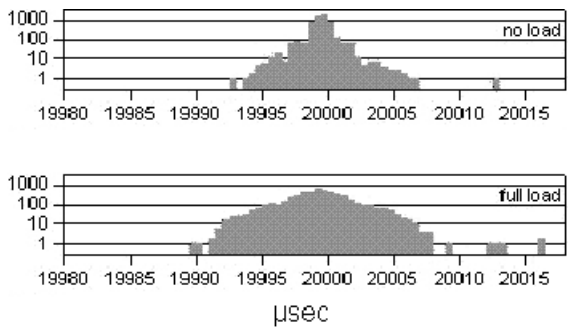


Figure 1-4

Représentation sur un système temps réel

Lorsque la charge est maximale, le système temps réel assure donc une variation de ± 0,2 Hz, alors que la variation est de ± 4 Hz dans le cas d'un système classique.

Cette expérience met en évidence l'importance des systèmes temps réel pour certaines applications critiques. En revanche :

- Un système temps réel aura un plus faible rendement, à matériel égal, qu'un système classique.

- L'utilisation d'un système temps réel pourra poser des problèmes de compatibilité matérielle car les pilotes de périphériques adaptés ne sont pas toujours fournis par le constructeur – en raison d'une trop faible diffusion.
- Pour la même raison, les API (*Application Programming Interfaces*, interfaces de programmation d'applications) et outils de développement d'un système classique seront souvent plus conviviaux que ceux d'un système temps réel.

Préemption et commutation de contexte

La notion de préemption intervient à deux niveaux :

- celui de la gestion du multitâche,
- celui du noyau du système.

Dans un système utilisant un multitâche préemptif, l'ordonnanceur pourra interrompre une tâche suite, par exemple, à la réception d'une interruption. Les vrais systèmes multitâches utilisent ce type de technologie, citons Unix, Linux, Windows NT. Dans un système multitâche non préemptif, la tâche ne peut être interrompue par le système ; elle doit directement demander l'intervention de l'ordonnanceur. On parle également de multitâche *collaboratif*. Le système Windows 3.11 est un multitâche collaboratif.

Le noyau (*kernel*) est le composant principal d'un système d'exploitation multitâche moderne. Dans un tel système, chaque tâche (ou processus) est décomposée en *threads* (*processus léger* ou *tâche légère*) ; ce sont des éléments de programmes, chacun étant capable d'exécuter une portion de code dans un même espace d'adressage. Chaque thread est caractérisé par un *contexte* local contenant la *priorité* du thread, ses variables locales ou l'état de ses registres. Le passage d'un thread à un autre est appelé changement de contexte (*context switch*). Ce changement de contexte sera plus rapide sur un thread que sur un processus car les threads d'un processus évoluent dans le même espace d'adressage, ce qui permet le partage des données entre les threads d'un même processus. Dans certains cas, un processus ne sera composé que d'un seul thread et le changement de contexte s'effectuera sur le processus lui-même.

Remarque

Dans le cas du système Linux, le même appel système `clone()` est utilisé pour créer un processus ou un thread. La primitive `fork()` de création de processus correspond en fait à `clone(0, SIGCLD|COPYVM)`.

Dans le cas d'un système temps réel, le noyau est dit *préemptif*, c'est-à-dire qu'un thread peut être interrompu par l'ordonnanceur en fonction du niveau de priorité, et ce afin de permettre l'exécution d'un thread de plus haut niveau de priorité. On peut ainsi affecter les plus hauts niveaux de priorité à des tâches dites *critiques* par rapport à l'environnement réel contrôlé par le système. La vérification des contextes à commuter est réalisée de manière régulière par l'ordonnanceur en fonction de l'horloge logicielle interne du système, ou *tick* système.

Dans le cas d'un noyau non préemptif, un thread sera interrompu uniquement dans le cas d'un appel au noyau ou d'une interruption externe. La notion de priorité étant très peu utilisée (excepté avec la commande `nice` dans le cas de Linux), c'est le noyau qui décide ou non de commuter le thread actif en fonction d'un algorithme complexe.

Résumé

On ne peut pas prévoir à l'avance le temps de réponse d'un système classique. Un système temps réel a en revanche un comportement prévisible, mais souvent un rendement moins élevé qu'un système classique, à configuration égale. Dans la majorité des cas, un système classique assorti d'une étude préalable de dimensionnement et de sécurisation permettra de satisfaire aux contraintes de temps réel « mou », et la programmation sur un tel système sera plus aisée.

Le tableau ci-après présente une synthèse de la comparaison entre un système à temps partagé et un système temps réel.

Tableau 1-1. Comparaison des critères temps réel et partagé

Critère	Temps partagé	Temps réel
Global	Avoir la meilleure capacité de traitement	Être prévisible
Temps de réponse	Doit être bon en moyenne	Doit être bon même dans le pire des cas, la moyenne n'a pas d'importance
Comportement à la charge	Confortable pour l'utilisateur	Stabilité et respect des contraintes de temps

Extensions Posix

La complexité des systèmes et l'interopérabilité omniprésente nécessitent une standardisation de plus en plus grande tant au niveau des protocoles utilisés que du code source des applications. Même si elle n'est pas obligatoire, l'utilisation de systèmes conforme à *Posix* est de plus en plus fréquente.

Posix est l'acronyme de *Portable Operating System Interface* ou interface portable pour les systèmes d'exploitation. Cette norme a été développée par l'IEEE (*Institute of Electrical and Electronic Engineering*) et standardisée par l'ANSI (*American National Standards Institute*) et l'ISO (*International Standards Organisation*).

Le but de Posix est d'obtenir la portabilité des logiciels au niveau de leur code source. Le terme de *portabilité* est un anglicisme dérivé de *portability*, lui-même réservé au jargon informatique. Un programme qui est destiné à un système d'exploitation qui respecte Posix doit pouvoir être adapté à moindre frais sous n'importe quel autre système Posix. En théorie, le portage d'une application d'un système Posix vers un autre doit se résumer à une compilation des sources du programme.

Remarque

Le concept de portabilité est très important d'un point de vue économique en termes de développement logiciel car le portage des applications génère souvent des frais énormes pour les éditeurs de logiciel, ce qui explique que certains logiciels existent uniquement pour les systèmes d'exploitation les plus répandus. Bob Scheifler, un des responsables du projet *X Window System*, a déclaré à ce sujet : « Il n'existe pas de logiciel portable, seulement des logiciels portés. »

Posix a initialement été mis en place pour les systèmes de type Unix mais d'autres systèmes d'exploitation comme Windows NT sont aujourd'hui conformes à Posix. Le standard Posix est divisé en plusieurs sous-standards dont les principaux sont les suivants :

- IEEE 1003.1-1990 : Posix partie 1 : Interface de programmation (API) système. Définition d'interfaces de programmation standards pour les systèmes de type Unix, connue également sous l'appellation ISO 9945-1. Ce standard contient la définition de ces fonctions (*bindings*) en langage C.
- IEEE 1003.2-1992 : interface applicative pour le s*hell* et applications annexes. Définit les fonctionnalités du shell et commandes annexes pour les systèmes de type Unix.
- IEEE 1003.1b-1993 : interface de programmation (API) temps réel. Ajout du support de programmation temps réel au standard précédent. On parle également de Posix.4.
- IEEE 1003.1c-1995 : interface de programmation (API) pour le multithreading.

Pour le sujet qui nous intéresse, la plupart des systèmes d'exploitation utilisables dans les technologies de l'embarqué sont conformes partiellement ou totalement au standard Posix. C'est le cas en particulier des systèmes temps réel LynxOS (*http://www.lynuxworks.com*) et QNX (*http://www.qnx.com*). Quant à Linux, sa conformité par rapport à Posix 1003.1b est partielle dans sa version standard et requiert l'application de modifications (ou *patch*) sur les sources du noyau. Il est cependant probable que la version standard sera conforme dans le courant de l'année 2002.

Définition de l'empreinte mémoire

Par définition, l'*empreinte* est la taille mémoire occupée par le système. Même si ce n'est pas une obligation, les systèmes embarqués ont en général une faible empreinte, et ce afin d'optimiser le système tant au niveau des coûts que de la sécurité de fonctionnement. La réduction de l'empreinte mémoire est la tâche principale d'un développeur de système embarqué car elle a un impact économique énorme sur l'industrialisation finale du produit. Plusieurs chapitres de la deuxième partie de l'ouvrage seront consacrés à ce problème.

Le tableau 1-2 donne une idée des empreintes mémoire, mémoires vive (RAM) et morte (ROM), en fonction du type d'application. Les valeurs données sont indicatives et des technologies spécifiques comme les disques mémoires (ramdisk) ou bien les systèmes de fichiers compressés (*compressed filesystem*) pourront modifier fortement le rapport RAM/ROM.

Tableau 1-2. Empreinte mémoire en fonction du type d'application

Produit	Serveur	Desktop	PC emb.	Emb. gros	Emb. moyen	Emb. typique	Profondément enfoui
RAM en Mo	128 ou +	128-32	64-16	32 à 8	8 à 2	4 à 0,1	Moins de 0,1
ROM en Mo	Plusieurs milliers	Plusieurs centaines	64 ou plus	32 à 8	8 à 2	2 à 0,5	0,5 à 0,1

Remarque

La notion de ROM citée ici l'est par opposition à celle de RAM. Tout ou partie de la mémoire ROM sera souvent constituée de mémoire *flash* réinscriptible.

Les deux premières colonnes ne concernent en général pas le monde des applications embarquées. Il faut cependant faire la différence entre l'empreinte mémoire du système d'exploitation et celle des données. Dans certains cas, on pourra par sécurité avoir un système d'exploitation réduit à quelques méga-octets (Mo) sur une mémoire morte et un disque dur, ou autre périphérique de stockage de masse de plusieurs dizaines ou centaines de giga-octets destiné à recevoir les données.

Les systèmes dits profondément enfouis (*deeply embedded*) sont souvent à la limite de l'utilisation d'un système d'exploitation par rapport à un logiciel embarqué dédié. Cependant, les applications de ce type sont le plus souvent embarquées dans des équipements portables de faible volume comme des téléphones mobiles qui nécessitent de plus en plus de fonctions de communications avancées, tout en conservant une taille mémoire raisonnable en rapport avec leur large diffusion. Des systèmes comme QNX ou eCos (voir pour eCos : *http://ecos.sourceware.org* et *http://www.ecoscentric.com/*) seront bien adaptés à ce type d'application.

Les langages utilisés

Pour des raisons évidentes de contraintes matérielles, le langage *assembleur* a longtemps été le choix de prédilection des technologies de l'embarqué car il permettait à la fois d'optimiser la taille du code généré mais aussi ses performances. La gestion d'un projet complexe écrit en assembleur est cependant difficile et l'évolution des performances du matériel et des compilateurs permet aujourd'hui de se tourner vers des solutions plus confortables. Les langages C et C++ restent aujourd'hui le choix favori des développeurs en partie à cause des liens privilégiés du langage C avec les standards Posix et les systèmes de type Unix. Historiquement, le langage C est également un langage permettant une programmation relativement proche du matériel, donc bien adaptée au logiciel embarqué.

La contrainte principale étant le plus souvent l'empreinte mémoire, la programmation en langage C nécessite d'utiliser le compilateur de la manière la plus efficace possible, et ce

en utilisant au maximum les options d'optimisation adaptées au matériel. Dans le cas du compilateur GNU *gcc* (*GNU C Compiler*), on notera en particulier les options suivantes `-O3`, `-O2`, `-O3`, selon le niveau d'optimisation, et `-Os` qui permet d'optimiser le code afin de minimiser sa taille. Des options spécifiques au processeur utilisé comme `-m386` ou `-mpentium` permettront également d'adapter le code au matériel utilisé.

Dans le cas de l'utilisation de systèmes de type Unix (on dit aussi *UNIX-like*), on pourra également employer d'autres langages de programmation ou langages de scripts plus appropriés dans certains cas. On citera en particulier le *shell-script*, langage de script d'Unix (ou *Bourne shell*, du nom de son concepteur Steve Bourne), qui associé à d'autres commandes pourra se révéler très utile dans l'écriture de procédures système. Le résultat sera le plus souvent plus facile à maintenir qu'un programme écrit en langage C et surtout ne nécessitera pas de recompilation avant exécution sur un autre système de type Unix. Des versions très fonctionnelles du Bourne-shell comme *ash* ou *msh* n'occupent respectivement pas plus d'une soixantaine et une trentaine de kilo-octets, ce qui est très admissible dans la majorité des systèmes. D'autres produits encore plus réduits, comme *BusyBox* que nous décrirons dans la deuxième partie, permettent d'intégrer des versions simplifiées d'un bon nombre de commandes Unix dans seulement 150 kilo-octets.

D'autres langages de scripts célèbres dans le monde Unix comme Perl, Tcl/Tk ou Python sont eux assez peu utilisés dans les environnements embarqués de par l'espace mémoire nécessaire à leur installation et à leur fonctionnement.

Tour d'horizon des systèmes existants

Nous allons terminer ce chapitre en effectuant un rapide tour d'horizon des principaux systèmes d'exploitation utilisés dans les environnements embarqués. Ce tour d'horizon n'inclut pas les systèmes à base de Linux qui seront décrits dans le chapitre suivant.

Remarque

Il est difficile de recenser les nombreux systèmes embarqués existants d'autant que certains industriels ont parfois développé leurs propres systèmes afin de satisfaire à des besoins ou des normes très particulières (cas de l'aéronautique).

VxWorks et pSOS

À tout seigneur tout honneur, VxWorks est aujourd'hui le noyau temps réel le plus utilisé dans l'industrie. Il est développé par la société Wind River (*http://www.windriver.com*) qui a également racheté récemment les droits du noyau temps réel *pSOS*, un peu ancien, mais également largement utilisé. VxWorks inclut en natif un support TCP/IP et une interface de programmation intégrant les *sockets*. Le gros inconvénient de ces deux systèmes est le coût important des licences. Il est également nécessaire d'utiliser un environnement de compilation croisé.

Rappel

Un environnement de compilation croisé permet de développer pour le système cible sur une autre machine dans un environnement plus confortable. On peut citer par exemple la disponibilité sous Linux d'un environnement de développement pour PalmOS, basé sur le compilateur GNU gcc.

QNX

Développé par la société canadienne QNX Software (*http://www.qnx.com*), QNX est un noyau temps réel de type Unix très intéressant. Il est parfaitement conforme à Posix, permet de développer directement sur la plate-forme cible et intègre l'environnement graphique *Photon*, proche de *X Window System*. Conscient de la percée de Linux dans le monde de l'embarqué, QNX Software s'en est rapproché en mettant à disposition la majorité des outils GNU sur la plate-forme QNX. Autre avantage non négligeable : QNX peut être utilisé gratuitement pour des applications non commerciales.

Il peut occuper une très faible empreinte mémoire et, de ce fait, peut être utilisé dans des environnements très réduits comme les téléphones portables GSM Timeport de chez Motorola.

µC/OS (micro-C OS) et µC/OS II

µC/OS, développé par le Canadien Jean J. Labrosse, est destiné à des environnements de très petite taille comme des micro-contrôleurs de type Motorola 68HC11. Il est maintenant disponible sur un grand nombre de processeurs et peut intégrer des protocoles standards comme TCP/IP (µC/IP). Il est utilisable gratuitement pour l'enseignement et de nombreuses informations sont disponibles sur *http://www.ucos-ii.com*.

Windows CE

Annoncé avec fracas par Microsoft comme le système d'exploitation embarqué *qui tue*, Windows CE et ses cousins comme Embedded Windows NT n'ont pour l'instant pas détrôné les systèmes embarqués traditionnels. Victime d'une réputation de fiabilité approximative, Windows CE est pour l'instant cantonné à l'équipement de nombreux assistants personnels (PDA, *Personal Digital Assistant*). Des informations sont disponibles sur *http://www.microsoft.com/windows/embedded*.

LynxOS

LynxOS est développé par la société LynuxWorks (*http://www.lynuxworks.com*) qui a récemment modifié son nom en raison de son virage vers Linux avec le développement de BlueCat. C'est un système temps réel conforme à la norme Posix.

Nucleus

Nucleus est développé par la société Accelerated Technology Inc. (*http://www.accelerated-technology.com*). C'est un noyau temps réel qui inclut une couche TCP/IP, une interface graphique (Graphix) ainsi qu'un navigateur web (WebBrowse) et un serveur HTTP (WebServ). Il est livré avec les sources et il n'y pas de *royalties* à payer pour la redistribution.

eCos

Acronyme pour *Embeddable Configurable Operating System*, eCos fut initialement développé par la société Cygnus, figure emblématique et précurseur de l'Open Source professionnel, acquise ensuite par la société Red Hat Software. C'est un système d'exploitation temps réel bien adapté aux solutions à très faible empreinte mémoire et profondément enfouies. Son environnement de développement est basé sur Linux et la chaîne de compilation GNU avec conformité au standard Posix et disponibilité de protocoles standards comme TCP/IP. Argument non négligeable, il est diffusé sous une licence proche de la GNU GPL (G*eneral Public licence*) et disponible en sources sur *http://ecos.sourceware.org*. La société eCosCentric fondée en 2002 par les responsables du projet eCos chez Red Hat fournit des versions professionnelles et du support (voir *http://www.ecoscentric.com*).

Le système eCos est largement utilisé dans l'industrie automobile, dans certaines imprimantes laser ou bien des produits multimédias comme le lecteur MP3 HitZip de chez Iomega. Il est disponible pour un grand nombre de processeurs comme les x86, PowerPC, ou SH3 Hitachi ou StrongARM.

En résumé

Les logiciels et systèmes embarqués sont toujours dédiés à un équipement.

Un équipement dédié à une tâche donnée est appelé *appliance* ou *serveur d'application*.

Les systèmes classiques utilisent le principe du *temps partagé*. Sa priorité est le confort de l'utilisateur. Un système temps réel doit en revanche respecter les contraintes temporelles, et ce quelle que soit sa charge. Son comportement doit être *prévisible*. La version standard du noyau Linux n'est pas temps réel.

L'empreinte mémoire d'un système embarqué peut varier de 64 Mo à 0,1 Mo. On parle alors de système *profondément enfoui*.

Les langages les plus répandus dans le monde de l'embarqué sont le C, le C++ et l'assembleur.

Il existe une multitude de systèmes d'exploitation à embarquer. Plusieurs déclinaisons de Linux existent d'ores et déjà pour les applications embarquées ; elles seront citées au chapitre suivant.

2

Linux comme système embarqué

Le chapitre précédent a décrit quelles étaient les caractéristiques d'un système embarqué, les différents types de systèmes ainsi que leurs applications. Le présent chapitre va s'attacher à détailler les nombreux avantages – mais aussi les quelques contraintes – inhérentes au choix de Linux comme système embarqué.

Contraintes des systèmes embarqués propriétaires

Le monde des systèmes embarqués est encore dominé par les éditeurs de solutions propriétaires comme le démontre la figure ci-après, qui donne la répartition des systèmes utilisés à la fin du XX^{e} siècle :

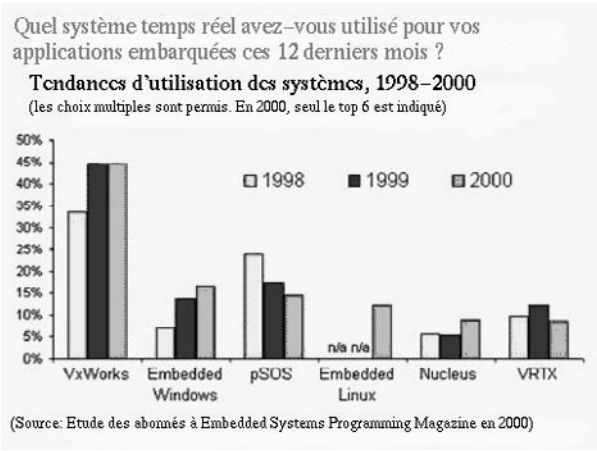


Figure 2-1

Répartition des systèmes

Ces systèmes, pour la majorité, souffrent cependant de quelques défauts forts contraignants pour les concepteurs d'équipement.

Ils sont souvent réalisés par des sociétés de taille moyenne qui ont du mal à suivre l'évolution technologique : le matériel évolue très vite – la durée de vie d'un processeur est de l'ordre de 12 à 24 mois ; il en est de même des standards logiciels et de plus en plus d'équipements nécessitent l'intégration de composants que l'on doit importer du monde des systèmes informatiques classiques ou du multimédia.

De ce fait, les coûts de licence et les droits de redistribution des systèmes sont parfois très élevés car l'éditeur travaille sur un segment de marché très spécialisé, une *niche* dans laquelle les produits commercialisés le sont pour leur fonction finale et non pour la valeur du logiciel lui-même. Contrairement au monde de la bureautique où la pression commerciale peut inciter l'utilisateur à faire évoluer son logiciel fréquemment – et donc à payer un complément de licence –, le logiciel embarqué est considéré comme un mal nécessaire, souvent destiné à durer plusieurs années, en conservant une compatibilité avec du matériel et des processeurs obsolètes.

Concernant ce dernier point, l'industriel qui utilise un système propriétaire prend le risque de voir disparaître l'éditeur ou même le système ainsi que le support technique qui va avec. Pour éviter au maximum ce genre de situation dramatique, les industriels paient souvent des sommes conséquentes afin de disposer de tout ou partie des sources du système. Si tel n'est pas le cas, ils devront utiliser des méthodes hasardeuses comme le *reverse engineering* (ingénierie inverse ou « rétrotechnique ») qui consiste à tenter de comprendre le fonctionnement du logiciel sans disposer des sources, et ce en effectuant un « désassemblage » des programmes.

Le coût de développement d'applications autour de systèmes propriétaires est souvent plus élevé car les outils de développement disponibles sur le marché du travail sont mal connus de la majorité des développeurs car non étudiés à l'université. Il est donc nécessaire de recruter du personnel très spécialisé donc rare... et cher. Les formations, très « pointues », autour de ces outils sont également onéreuses, ce qui oblige l'éditeur à pratiquer des coûts élevés pour compenser le manque d'effet de masse. Tout cela implique un ensemble de spécificités contraignantes pour la gestion globale des outils informatiques de l'entreprise.

Les avantages de l'Open Source

Le concept d'Open Source a été introduit dans l'avant-propos. Les trois points suivants de la définition du logiciel Open Source sont fondamentaux dans le cas du logiciel embarqué :

- redistribution sans royalties ;
- disponibilité du code source ;
- possibilité de réaliser un développement dérivé de ce code source.

Le premier point règle le problème économique des droits de redistribution ou royalties, très contraignant dans le cas d'un système distribué à grande échelle. La disponibilité du code source est encore plus fondamentale car elle est la base de la conception d'un logiciel de qualité et surtout maintenable dans le temps.

Si un bogue est détecté dans le système sur lequel sont développées les applications, la disponibilité du code source permettra à coup sûr de corriger le problème, à supposer que l'on dispose de la compétence adéquate. Sur ce point, le fait d'utiliser des logiciels largement répandus comme Linux augmente les chances de pouvoir trouver assez facilement cette compétence.

Si le système doit « vivre » plusieurs années, il sera toujours possible grâce à l'Open Source d'en réaliser une image complète ainsi que de tous les outils de développement associés, de manière à pouvoir générer à tout moment un nouveau système cible sur une plate-forme compatible. Ce critère de pérennité est extrêmement important dans un processus industriel.

La disponibilité du code source facilite la compréhension du système et donc augmente la qualité et la stabilité des applications développées pour ce dernier. La documentation des logiciels Open Source largement répandus comme Linux est souvent de très bonne qualité car elle a pu bénéficier d'un gros travail collaboratif. Même si – tradition et statistique obligent – elle est souvent plus à jour en langue anglaise, la diffusion publique de l'information permet également de disposer de versions internationales. Le projet *LDP* (*Linux Documentation Project*) disponible à partir du site *http://www.linuxdoc.org* en est un excellent exemple.

Le dernier point concernant la facilité de compréhension est vérifié quel que soit le type de développement logiciel, et ce indépendamment du côté Open Source. L'exemple le plus frappant est le navigateur Internet Explorer de Microsoft. Sans être un grand admirateur de Microsoft, il faut admettre que leur navigateur, développé par des équipes ayant une connaissance intime de Windows, est aujourd'hui plus performant que son rival Netscape Communicator. Cela ne signifie pas forcément que les développeurs de Microsoft sont meilleurs que ceux de Netscape, mais plutôt que la disponibilité des sources est un atout majeur pour la qualité des développements.

Même s'il est excessif d'affirmer qu'un logiciel Open Source est toujours de meilleure qualité qu'un logiciel « fermé », il est important de noter que le fait de diffuser les sources oblige le développeur à soigner ce code car il va ainsi être jugé par ses pairs. Tout vrai développeur de logiciel a un côté artiste et créatif qui provoque chez lui un sentiment d'excitation et de fierté lorsqu'un affichage graphique est plus harmonieux ou un algorithme de calcul plus optimisé. La notion d'Open Source est donc une bonne garantie de qualité, sans oublier que la publication des sources facilite d'autant l'amélioration rapide du code qui sera visible par des milliers d'autres programmeurs.

Il est fondamental de faire la différence entre le logiciel gratuit (*freeware*) et le logiciel Open Source communément appelé *logiciel libre* en français, bien que la traduction de *free* par *gratuit* conduise souvent à des confusions. Il existe de nombreux logiciels gratuits disponibles sur Internet dont les sources ne sont pas disponibles. Il peut aussi arriver

que les sources d'un logiciel gratuit soient disponibles pendant un certain temps mais les licences de ces logiciels, souvent peu précises, ne garantissent pas en général leur disponibilité, ce qui est un risque majeur dans un processus industriel.

La plupart des logiciels Open Source sont eux régis par des licences très structurées dont la plus célèbre est la GPL (*General Public Licence*), explicitée dans l'avant-propos. Cette licence est statistiquement la plus utilisée dans l'environnement Linux car elle permet de garantir le mieux possible la disponibilité permanente des sources d'un logiciel libre.

Contrairement à certaines légendes, l'utilisation de la GPL ou de la LGPL (*Lesser* GPL) est parfaitement compatible avec l'approche industrielle. Si tel n'était pas le cas, il ne serait pas possible de diffuser des versions Linux d'applications commerciales autres qu'Open Source. La LGPL permet la diffusion de code propriétaire – donc sans les sources – lié à des bibliothèques GPL comme la glibc (GNU libc). Même si cette possibilité ne satisfait pas les « jusqu'au-boutistes » du logiciel libre, elle a l'avantage d'avoir permis l'introduction progressive de Linux dans l'industrie, chose qui n'aurait pas été possible si l'on avait proposé aux industriels de remplacer leurs applications favorites par des équivalents, certes Open Source, mais à l'époque parfaitement inconnus du monde industriel.

La GPL n'est cependant pas la seule licence communément utilisée dans le monde du logiciel libre car elle est parfois considérée par certains comme trop contraignante. Les licences BSD (*Berkeley Software Distribution*) et MIT (*Massachusetts Institute of Technology*) sont également largement utilisées, en particulier pour l'interface graphique *X Window System* qui utilise la licence MIT. La différence principale avec la GPL est la non-obligation de rediffusion systématique (ou *copyleft*) des sources des applications dérivées de sources sous licence BSD ou MIT. L'énoncé des licences BSD et MIT est également beaucoup plus simple, quelques lignes au lieu de plusieurs pages pour la GPL et la LGPL.

The MIT licence

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

La liste des principales licences Open Source est disponible sur le site *http://www.opensource.org/licenses*.

Et les quelques contraintes...

L'utilisation de Linux et des logiciels libres peut cependant entraîner quelques contraintes dans un environnement industriel. En premier lieu, se pose le problème de la crédibilité de l'Open Source par rapport aux éditeurs de logiciels classiques. Le problème ne vient pas des ingénieurs qui sont la plupart du temps les premiers à valider la supériorité technique des solutions Open Source. L'Open Source a plutôt une approche communautaire qui peut provoquer la méfiance des décideurs pour lesquels ce mot a une connotation anti-libérale, et peut être interprété comme anti-profit. L'ingénieur ou l'équipe technique qui choisira un logiciel libre par rapport à une solution propriétaire pourra être en opposition avec sa hiérarchie, ce qui est rarement agréable. On ne licenciera jamais un directeur informatique pour avoir fait le choix d'un produit commercial qui a pignon sur rue, même si le logiciel connaît des problèmes d'installation ou de fonctionnement. Le choix d'une solution Open Source peut être beaucoup plus risqué politiquement car à la première difficulté les détracteurs ne manqueront pas de ricaner en disant : « C'est gratuit donc ça n'est pas fiable. ».

La rumeur et la méconnaissance des licences sont bien entendu entretenues par les éditeurs pour lesquels l'Open Source constitue une menace. Certains membres de la presse spécialisée, pour lesquels les éditeurs de solutions propriétaires constituent la principale clientèle d'annonceurs, furent dans le passé souvent peu enclins à démontrer la supériorité des solutions Open Source. La logique est économique : un gros éditeur pourra payer des spécialistes du marketing et des relations de presse, la même approche ne pourra pas être possible pour un projet Open Source.

En second lieu, se pose le problème important du support technique qui est fortement ancré dans les esprits. Par essence, le logiciel libre est livré *en l'état* (*as is*) et ce sans aucune garantie de bon fonctionnement. Si l'utilisateur désire de l'assistance, il devra utiliser les canaux traditionnels des documentations et FAQ – *Frequently Asked Questions*, parfois traduit en français par *Foire aux questions* – disponibles sur Internet, des groupes de discussion (*newsgroups*) ou le cas échéant des courriers électroniques échangés directement avec les auteurs. Ces méthodes sont techniquement très efficaces car un problème fréquent sur un logiciel libre est forcément référencé quelque part sur Internet. Durant ma longue expérience du logiciel libre, je ne peux pas citer un seul problème que je n'ai pas pu résoudre *via* la source directe qu'est Internet. Cette logique est cependant une approche d'ingénieur et celle d'un juriste ou d'un chef d'entreprise sera différente car ces derniers chercheront plutôt une responsabilité légale au problème même si cela ne le corrige pas !

Le point crucial du support est la pierre angulaire du modèle économique des sociétés gravitant autour du logiciel libre. Puisqu'elles ne peuvent faire de profit sur la vente de

licence, le profit sera réalisé sur le support associé au produit. Les gros éditeurs de distribution Linux comme Red Hat ou Mandriva (ex-Mandrake) suivent ce modèle : la distribution est disponible sur Internet ou bien sur des CD-Rom gratuits qui contiennent les mêmes paquetages logiciels qu'une version payante mais sur lesquels l'éditeur ne délivrera aucun support technique, ni documentation « papier ». La valeur ajoutée de la version payante se fera sur la présentation du produit et le support ou service associé.

D'autres sociétés sont exclusivement axées sur le support technique et le service, sans être éditrices de logiciel. Cette approche apporte une plus grande souplesse dans le choix des composants Open Source car cela permet de puiser les composants dans diverses distributions, voire dans des projets non édités. Cette démarche est plus difficile pour un éditeur de distribution qui aura la contrainte d'utiliser exclusivement ses produits, du moins dans certains domaines. Même si l'existence de ces sociétés est un grand pas en avant, leur jeunesse et leur assise financière parfois précaire peuvent gêner les industriels. À la différence des solutions propriétaires, une solution Open Source n'est cependant jamais définitivement perdue même si la société qui la supporte n'existe plus. La disponibilité des sources est la *meilleure* garantie de pérennité.

Enfin, parmi les contraintes existantes, il faut mentionner la complexité et la multiplicité des licences Open Source. Celles-ci sont en nombre relativement important – plus de vingt licences courantes répertoriées sur *http://www.opensource.org* – et leur approche peut être déroutante pour les juristes habitués aux licences classiques. La licence GPL s'étale sur 17 pages et il faut être un peu initié afin d'en saisir toutes les subtilités.

Dans le cas des systèmes embarqués, l'exemple classique de contrainte inhérente à la GPL est l'impossibilité de mixer le code GPL original du noyau Linux avec du code propriétaire. Si vous développez un pilote de périphérique à partir du source d'un pilote existant (donc sous GPL), la licence vous oblige à diffuser le source du pilote. Même si vous écrivez entièrement un pilote de périphérique, la licence GPL vous oblige à diffuser son source si vous désirez le lier *statiquement* au noyau original sous GPL. La solution élégante consiste donc à utiliser systématiquement les pilotes sous formes de modules dynamiques non liés à la partie statique du noyau. Cette technique est décrite dans le chapitre 4.

Dans le cas de diffusion d'applications propriétaires sous Linux, la LGPL (*Lesser* GPL) permet de diffuser une application utilisant des composants sous LGPL, comme la glibc (GNU libc, bibliothèque principale utilisée par tous les applicatifs Linux). L'extension de la GPL à la LGPL fut la condition *sine qua non* pour la disponibilité d'applications commerciales classiques sur la plate-forme Linux.

Une dernière contrainte concerne les problèmes de compatibilité ascendante. Pour des raisons commerciales, les éditeurs sont le plus souvent tenus de respecter la compatibilité de leur version courante avec des versions très anciennes. Le cas le plus célèbre est la compatibilité des systèmes Microsoft Windows avec l'ancien système MS-DOS, conservée jusqu'à la version Windows 98. Les développeurs de projets Open Source n'ont pas cette contrainte et la compatibilité ascendante du produit n'est pas garantie. Il faut cependant

ajouter un bémol à ce dernier point. La disponibilité des sources complètes du système permet à tout moment de travailler sur un produit, même très ancien, et d'y ajouter certains composants actuels en effectuant un portage « rétro-actif » (*backport*), d'autant que Linux est souvent beaucoup mieux structuré que d'autres systèmes.

Pourquoi Linux est-il adapté à l'embarqué ?

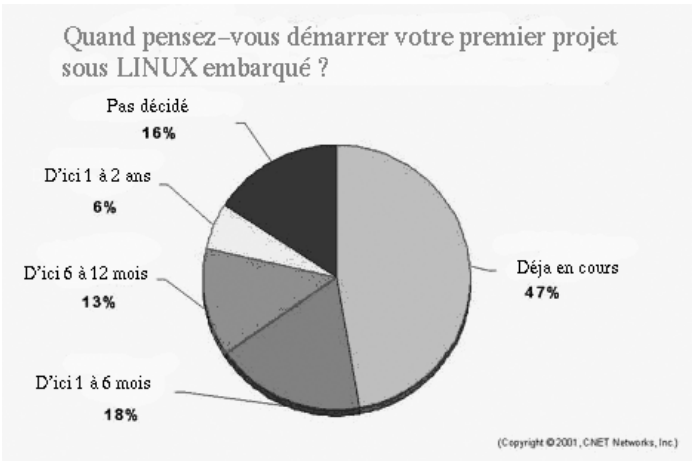
Si les systèmes propriétaires dominent encore le marché de l'embarqué en ce début de XXIe siècle, une étude de marché réalisée en 2000 par VDC (Venture Development Corporation) prévoit un marché Linux embarqué de 305 millions de dollars en 2005 contre seulement 28 millions en 2000 et 55 millions en 2001. De même, si 59 % des industriels affirment début 2001 n'avoir jamais utilisé Linux pour une application embarqué, 19 % répondent l'avoir utilisé sur un projet et 22 % affirment l'avoir utilisé sur de multiples projets (source CNET Networks Inc.).

Le graphique ci-après indique la répartition de l'utilisation de Linux sur des délais de projets variant de zéro (projets actuels) à deux ans. Là encore, 47 % des industriels interrogés affirment avoir actuellement un ou plusieurs projets utilisant Linux comme système embarqué (source CNET Networks Inc.).

Remarque

Suite à l'étude *Embedded Linux Market Survey* réalisée par LinuxDevices.com, en 2003/2004, des statistiques plus récentes sont disponibles à l'adresse *http://www.linuxdevices.com/news*. Nous vous encourageons donc à consulter les dernières versions en ligne.

Figure 2-2
Répartition des projets



Dernier élément, parmi les 58 % des industriels qui ont choisi Linux, 20 % déclarent que c'est en raison de la disponibilité des sources, 19 % pour le coût et 18 % pour la fiabilité. Le graphe présenté ci-après donne la répartition complète des motivations.

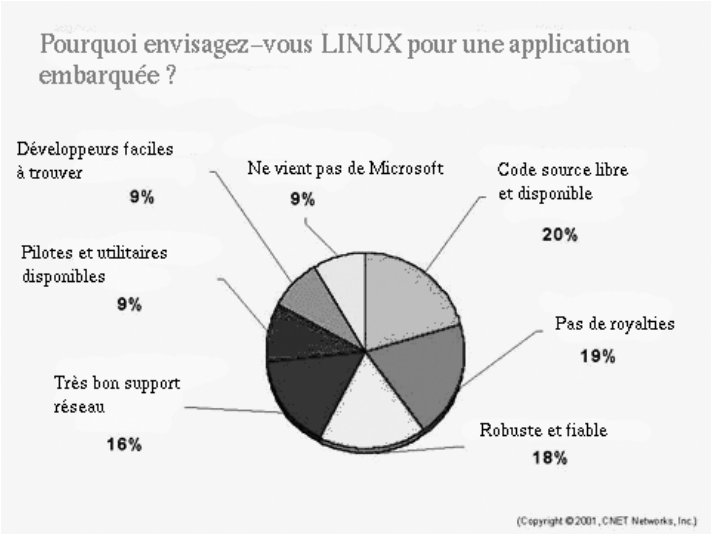


Figure 2-3
Répartition des motivations

Fiabilité

Linux est réputé pour sa fiabilité et l'on peut affirmer que cette réputation n'est pas usurpée. En dix ans d'utilisation de Linux, je peux compter sur les doigts de la main les cas de « plantage » inexpliqués. Le fameux *kernel panic* (erreur fatale du noyau) tant redouté par les développeurs est un animal rare, presque autant que le Yéti ou le monstre du Loch Ness. Une erreur de ce type s'explique toujours par un problème matériel ou une erreur de programmation dans un pilote de périphérique. Ces derniers travaillant dans un espace privilégié appelé « espace noyau » (*kernel space* ou *kernel level*), ils sont les seuls à pouvoir provoquer ce type d'erreur. Les autres applications travaillent dans l'espace dit utilisateur (*user space* ou *user level*) et n'ont pas les droits nécessaires à la génération de ce type d'erreur.

Attention

Linux dispose cependant de fonctions spéciales (comme `iopl` ou `ioperm`) permettant à un programme de l'espace utilisateur de s'approprier des droits d'accès à des ressources normalement réservées au noyau et susceptibles de générer des plantages du système. L'utilisation de ces fonctions doit être réservée à des cas extrêmes ou des situations temporaires et il est toujours plus prudent de développer un pilote de périphérique. Notez que ces fonctions ne sont par portables vers d'autres systèmes.

La fiabilité de Linux est démontrable au moyen de la commande **`uptime`** qui permet d'afficher le temps d'activité du système depuis le dernier redémarrage. La valeur du

temps d'activité donne lieu à des concours, et des valeurs de plusieurs mois sont monnaie courante. La couche TCP/IP de Linux, au cœur de nombreuses applications embarquées communicantes, est également d'une grande fiabilité.

Faible coût

La contrainte économique est bien évidemment très importante dans le cas du développement d'un système embarqué. Linux est non seulement exempt de royalties mais les outils de développement sont également disponibles sous GPL. Le seul effort financier nécessaire à l'adoption de Linux se situe sur la formation – souvent indispensable – et le support technique.

La période difficile ayant suivi l'engouement pour la nouvelle économie peut être de ce fait une chance pour Linux car il est beaucoup moins coûteux de réaliser une étude avec Linux qu'avec un autre système fonctionnant sur une base de royalties.

Attention

Nous ne saurions trop mettre en garde le lecteur contre la tentation du « tout gratuit » sous Linux. Le fait de recourir à des prestations externes pour la formation et l'assistance au développement d'un projet Linux embarqué peut faire gagner un temps précieux et permettre un transfert technologique efficace en vue des projets suivants. Plus généralement, cette approche du « tout gratuit » peut mettre en péril l'industrie du logiciel libre exclusivement basée sur le service et le support. Le fait de profiter des énormes avantages de l'Open Source implique une certaine déontologie qui se révèle être un avantage à moyen et long terme. Les statistiques semblent rassurantes sur ce point puisque 68 % des utilisateurs sont disposés à payer un support technique contre 13 % qui n'y sont pas disposés et 19 % qui sont indécis (source CNET Networks Inc.).

Performances

Les performances de Linux ne sont plus à prouver. De nombreux tests comparatifs (*benchmarks*) entre Linux et d'autres systèmes concurrents comme Windows NT on démontré la supériorité de Linux. Des résultats de tests sont disponibles en particulier sur le site du *Linux Test Project* (*http://ltp.sourceforge.net*). Cependant, les benchmarks ne sont pas forcément le meilleur critère car ils sont souvent réalisés dans des conditions particulières, parfois éloignées des situations réelles. Pour se rendre compte des capacités de Linux, le mieux est encore de le tester soi-même. Détail intéressant pour les applications embarquées, c'est dans des situations de faibles performances matérielles que Linux se révèle le plus efficace par comparaison à d'autres systèmes.

Portabilité et adaptabilité

La portabilité est également un des points forts de Linux. Même si le premier courrier de Linus Torvalds en 1991 annonçait clairement que Linux ne tournerait jamais sur autre chose que les processeurs x86, le fait est qu'il s'est pour une fois lourdement trompé. Linux est aujourd'hui porté sur un très grand nombre de processeurs et d'architectures

matérielles, y compris des processeurs de faible puissance, comme le démontre le projet µClinux (*http://www.uclinux.org*). Même si le processeur ou l'architecture que vous désirez utiliser ne figurent pas dans la liste des portages actuels, l'énorme base de connaissance disponible facilitera le travail et vous pourrez souvent vous inspirer de travaux déjà réalisés. De même, sachant que Linux est de plus en plus introduit dans les universités et organismes d'éducation en tout genre, les compétences seront beaucoup plus faciles à trouver que sur d'autres systèmes à la réputation plus confidentielle.

La structure modulaire de Linux héritée de l'architecture Unix est également un de ses gros avantages. La structure du système est stricte et clairement définie, et il sera aisé de le configurer de manière à trouver une correspondance exacte avec les besoins dans les limites des contraintes matérielles, quitte à ajouter des composants plus tard. Si on souhaite par exemple ajouter le support d'un protocole réseau comme PPP (*Point to Point Protocol*), il suffira d'ajouter au système le module noyau du support PPP ainsi que le client de connexion, ces derniers étant validés depuis longtemps sur les versions complètes du système. Il ne faut jamais oublier que les développements Linux ne sont pas limités à ceux d'un système embarqué, ce qui est un gros avantage, tant en termes de rapidité de disponibilité des composants que pour ce qui est de leur qualité.

Ouverture

De par sa conception Open Source, Linux est ouvert. Ce concept d'ouverture se situe à deux niveaux.

Le premier niveau concerne l'interopérabilité de Linux avec d'autres systèmes d'exploitation. Certains systèmes comme les différentes versions de Windows ont une approche hégémonique de l'informatique, c'est-à-dire qu'ils considèrent par défaut qu'ils sont les seuls acteurs d'une configuration informatique complète. C'est bien entendu absurde et l'utilisateur en fait les frais ; nous citerons par exemple la procédure d'installation Windows 9x qui détruira invariablement le secteur de démarrage (*boot sector*) du disque même si un autre système comme Linux est déjà installé. Il y a pire, par exemple lorsque la même procédure d'installation peut s'allouer autoritairement tout le volume du disque si l'utilisateur n'a pas pris la précaution d'estampiller la première partition comme étant de type FAT32 qui est un des formats de système de fichier de Windows.

Au contraire, Linux a une approche collaborative, ce qui signifie qu'il va s'intégrer dans un environnement existant de manière à optimiser les performances globales du système informatique et donc faciliter la vie de l'utilisateur. Les exemples en sont nombreux et on peut en citer quelques-uns :

- Linux permet depuis le début de lire et écrire les données présentes sur une partition Windows (FAT ou FAT32). Le contraire n'est possible qu'après installation de logiciels tiers.
- Linux peut s'installer sur n'importe quelle partition de n'importe quel disque du système. Cela n'est pas possible pour certaines versions grand public de Windows qui ne fonctionnent qu'à partir de la première partition du premier disque.

- Linux inclut des programmes de démarrage comme LILO ou GRUB qui permettent de démarrer alternativement sur l'un des systèmes présents sur la machine. Cela n'est possible que sur certaines versions de Windows.
- Linux supporte depuis le début le protocole réseau de Windows appelé SMB ou CIFS. Il suffit pour cela d'installer le logiciel Open Source SAMBA (*http://www.samba.org*) livré avec Linux. Le support du protocole NFS, équivalent Unix de SMB, n'est disponible qu'en ajoutant à Windows un logiciel tiers.

Le second niveau d'ouverture concerne l'adoption dans Linux des nouvelles technologies, pour peu que celles-ci constituent des standards ouverts. Le fait que Linux soit totalement Open Source facilite l'adaptation et le test de nouveaux protocoles car tous les secrets du système sont à portée de main, du moins si l'on sait où les chercher. La plateforme Linux constitue donc un choix avantageux pour les développeurs et les scientifiques d'autant que les expériences de développement sont le plus souvent partagées sur Internet. Nous citerons à ce titre les plates-formes collaboratives SourceForge (*http://www.sourceforge.net*) qui héberge à ce jour environ 30 000 projets Open Source et le plus récent GNU Savannah (*http://savannah.gnu.org*) qui en héberge « seulement » 400.

Dans quels cas Linux peut-il être inadapté ?

Il peut exister des cas dans lesquels Linux sera inadapté. Si le système à embarquer nécessite uniquement des fonctions de base n'incluant pas de support réseau ni de multitâche et si cet équipement n'est pas destiné à évoluer, il n'est pas forcément intéressant d'utiliser un système aussi riche que Linux. En effet, un noyau Linux occupera au minimum 400 kilo-octets en version compressée et il sera difficile de tomber en dessous du méga-octet pour un système minimal fonctionnel. Concernant la mémoire vive, il ne faut pas espérer faire fonctionner correctement le noyau standard dans moins de 4 méga-octets.

Si vos contraintes matérielles ne sont pas compatibles, oubliez Linux et rabattez-vous sur d'autres systèmes comme eCos ou µC/OS cités dans le chapitre précédent, ou au pire sur un logiciel dédié développé par vos soins. Soyez cependant attentif à ne pas développer le système à trop court terme ; un système Linux est très évolutif et pourra suivre les évolutions technologiques de votre produit pendant longtemps. L'ajout d'un nouveau protocole à un système « maison » est souvent une tâche longue et difficile.

L'utilisation de la GPL/LGPL peut également se révéler contraignante ou bien heurter la sensibilité de votre hiérarchie ou la vôtre. Dans ce cas, il se peut que vous soyez contraints d'utiliser une solution propriétaire ou bien une autre solution Open Source basée sur un autre type de licence, par exemple la licence BSD. Des industriels utilisent FreeBSD (*http://www.freebsd.org*) comme système embarqué entre autres pour des raisons de fâcherie avec la GPL. FreeBSD est un excellent système, au moins aussi performant que Linux, mais qui dispose d'une moins grande notoriété en dépit de son plus grand âge. Méfiez-vous cependant de contraintes liées à la disponibilité des pilotes de périphériques ou alors armez-vous de patience et de bons collaborateurs !

La conformité par rapport à certains standards industriels (comme les standards de l'aéronautique) peut être difficile à valider. La question n'est pas technique mais, du fait de la diversité des sources d'information et de diffusion des composants Linux, il existe rarement une société unique qui puisse décider de financer une étude de conformité. Si la conformité intéresse un faible nombre d'applications, les modifications nécessaires auront peu de chances d'être intégrées à l'arborescence officielle et si ces modifications sont finalement réalisées par une société ou un groupe d'individus, il y aura perte de compatibilité des sources par rapport à la version officielle du noyau.

Indépendamment des applications embarquées, la multiplication des sources d'informations et des standards, qui de ce fait n'en sont pas, est à la fois un avantage et un inconvénient de Linux. Il n'y a pas de voix unique, il existe toujours des partisans de GNOME, de KDE, ou d'autres bureaux graphiques moins connus, donc des utilisateurs de GTK et de Qt, les bibliothèques à la base de ces deux bureaux. Il s'ensuit une grande liberté mais aussi un sentiment de cafouillage pour les non-initiés – un peu à l'image du « bazar » qui se tient face à la cathédrale, intéressant mais parfois difficile à intégrer.

Les systèmes embarqués basés sur Linux

Il existe d'ores et déjà un grand nombre de distributions et composants embarqués basés sur Linux dont la majorité est constituée de projets Open Source.

MontaVista Linux

Développé par la société Monta Vista (*http://www.mvista.com*), MontaVista Linux est le leader des solutions Linux embarqué commerciales. MontaVista est à l'origine des modifications du noyau Linux afin d'améliorer la préemption de ce dernier et donc les fonctionnalités de temps réel « mou ». De nos jours, MontaVista met plutôt en avant la liste très fournie des processeurs supportés par ses outils de compilation (voir *http://www.mvista.com*).

BlueCat Linux

Ce produit est édité par LynuxWorks, créateur et éditeur du système temps réel LynxOS. La nouvelle version BlueCat Linux 5.0 est basée sur le noyau 2.6 et profite donc de la fonction de noyau préemptif de ce dernier. LynuxWorks annonce également que les exécutables développés sous BlueCat sont compatibles binaires avec leur système temps réel dur propriétaire LynxOS (voir *http://www.lynuxworks.com/products/bluecat/bluecat.php3*).

µClinux

µClinux est un portage du noyau Linux pour micro-contrôleurs et processeurs dépourvus de MMU (*Memory Management Unit*). De ce fait, il n'y a pas de protection mémoire d'un programme à l'autre et un programme peut planter un autre programme ou le noyau lui-même. Très ouvert, le système µClinux est disponible pour un grand nombre de

processeurs comme le ColdFire Motorola, la famille 68xxx, ARM, Intel i960, Axis ETRAX. Les nouvelles versions du noyau 2.6 sont très rapidement intégrées dans le projet.

La page du projet est disponible sur *http://www.uclinux.org*. Le projet est sponsorisé par la société Arccturus Networks et un support commercial est disponible sur *http://www.uclinux.com*.

RTLinux

RTLinux est un projet Open Source qui a pour but d'ajouter à un noyau Linux standard un noyau temps réel préemptif et à priorités fixes. Ce petit noyau temps réel, chargeable dynamiquement dans le système, considère le noyau Linux comme la tâche de plus faible priorité. RTLinux a été initialement développé au département d'informatique de l'Institut des mines et de technologie du Nouveau-Mexique par Victor Yodaiken et Michael Barabanov.

Le produit ne peut plus vraiment être considéré comme un projet Open Source, puis la version GPL de RTLinux n'évolue quasiment plus par rapport à la version commerciale éditée par FSMlabs (*http://www.fsmlabs.com*), société fondée par les auteurs de RTLinux.

RTAI

RTAI utilise un principe similaire à RTLinux (principe de double noyau). À l'origine RTAI fut développé à partir d'une version modifiée de RTLinux mais de nos jours c'est un projet très dynamique, performant et totalement indépendant de RTLinux. À la différence de son homologue commercial, RTAI est un projet entièrement libre.

ELDK

Le projet ELDK (*Embedded Linux Development Toolkit*) est maintenu par la société allemande Denx Software (*http://www.denx.de*). Ce produit d'excellente qualité permet le développement du logiciel en développement croisé depuis un PC Linux x86 vers les architectures PowerPC (PPC) ou ARM. ELDK fournit également une image de système utilisable par NFS (*Network File System*). ELDK est décrit en détails au chapitre 11 de cet ouvrage.

PeeWee Linux

Ce projet, disponible sur *http://www.peeweelinux.org*, est basé sur la distribution Red Hat 6.2. Il utilise une version standard du noyau 2.2 et n'inclut pas de fonctionnalités temps réel. Son principal attrait est l'existence d'un utilitaire de construction du système cible permettant de choisir les composants de manière assez conviviale (interface similaire à la configuration du noyau Linux). Il supporte les mémoires flash de type DiskOnChip ou bien les périphériques classiques IDE ou SCSI. Le site web du projet permet de télécharger une image du CD-Rom de la distribution. Ce projet faisait l'objet d'une présentation détaillée dans la première édition de cet ouvrage mais il est à ce jour obsolète et le chapitre 11 en question est remplacé par une description des environnements de compilation croisée.

Tableau 2-1. Récapitulatifs des systèmes embarqués

Nom	Éditeur	Open Source	Temps réel	URL	Remarques
VxWorks	WindRiver	Non	Oui	http://www.windriver.com	Certainement le leader du marché
pSOS	WindRiver	Non	Oui	http://www.windriver.com	Ancien mais très répandu
QNX	QNX	Partiel-lement	Oui	http://www.qnx.com	Basé sur UNIX, pas mal de composants Open Source, assez répandu, bonnes performances
µC/OS	Micriµm	Non	Oui	http://www.ucos-ii.com	Pour les micro-contrôleurs
Windows CE	Microsoft	Non	Oui	http://www.microsoft.com/windows/embedded	
LynxOS	LynuxWorks	Non	Oui	http://www.lynuxworks.com	Édite également BlueCat, basé sur Linux
Nucleus	Accelerated Technology	Oui	Oui	http://www.accelerated-technology.com	Pas de royalties
eCos	Red Hat	Oui	Oui	http://www.ecos.sourceware.org	Utilisable pour de très faibles empreintes mémoire, environnement de développement croisé Linux disponible
PeeWeeLinux	Aucun	Oui	Non	http://www.peeweelinux.org	Obsolète car fondé sur Red Hat 6.2, noyau 2.2
RTLinux	FSM Labs	Oui	Oui	http://www.fsmlabs.com	Existe en version GPL et version «pro», attention au brevet logiciel (patent), temps réel «dur»
RTAI	Aucun	Oui	Oui	http://www.rtai.org	Proche de RTLinux mais non ommercial
TUXIA	TUXIA	Oui	Non	http://www.tuxia.com	Dédié «Internet Appliance»
Red Hat Embedded Linux	Red Hat	Oui	Non	http://ww.redhat.com/embedded	
µClinux	Aucun	Oui	Oui	http://www.uclinux.org	Pour micro-contrôleurs sans MMU, sponsorisé par Arcturus Networks. Existe en noyau 2.4 et 2.6
Embedix	Lineo	Oui	Non	http://www.lineo.com	Utilisé dans le PDA Zaurus de SHARP
Montavista Linux (Hard Hat)	Monta Vista	Oui	Oui	http://www.mvista.com	Fondé sur des patches «préemptifs» du noyau Linux, pas de temps réel «dur»
ELDK	Denx Sofware	Oui	Non	http://www.denx.de	Environnement de développement croisé x86 vers ARM ou PowerPC

Quelques exemples de produits utilisant Linux

Linux est présent dans divers secteurs de l'industrie grand public. On peut citer :

- les assistants personnels ou PDA (*Personal Digital Assistant*) ;
- les consoles multimédias et tablettes Internet (*set top boxes* et *web pads*) ;
- les magnétoscopes numériques.

Dans des domaines plus professionnels, on peut citer :

- les routeurs ;
- les équipements de téléphonie ;
- les caméras IP.

Les PDA

Un des premiers modèles de PDA sous Linux fut le YOPI, développé par le coréen Samsung. Le système utilise un processeur StrongARM cadencé à 206 MHz avec un minimum de 16 méga-octets de RAM et 32 méga-octets de mémoire flash. Au niveau logiciel, le YOPI utilise le système graphique standard X Window System optimisé pour la circonstance. Contrairement à d'autres PDA pour lesquels il existe des versions de Linux remplaçant le système initial, le YOPI est conçu pour Linux.

Figure 2-4
Le YOPI de Samsung

Plus récemment, Sharp a mis sur le marché le ZAURUS, un PDA utilisant une architecture similaire mais dont l'interface est basée sur Java. Le système utilise un processeur StrongARM cadencé à 206 MHz avec un minimum de 16 méga-octets de RAM et 32 méga-octets de mémoire flash. Il dispose de nombreuses extensions et d'un petit clavier intégré.

Figure 2-5
Le ZAURUS de SHARP

On ne peut pas ne pas citer le célèbre iPAQ, produit phare développé par Compaq. Il n'est pas à ce jour diffusé par Compaq en version Linux mais avec Windows CE. Un portage de Linux et de l'interface graphique X Window System est cependant disponible comme l'indique la figure 2-6 ci-après.

Figure 2-6
iPAQ sous Linux et X

Plus généralement, les différents travaux concernant le portage de Linux et le développement d'applications Linux sur les PDA sont accessibles sur le site *http://www.handhelds.org*.

Les consoles multimédias et tablettes Internet

Ericsson a développé une tablette Internet utilisant une connexion sans fil (BlueTooth). Le système est architecturé autour d'un StrongARM cadencé à 206 MHz, 32 méga-octets de RAM et 32 méga-octets de mémoire flash. Il utilise la bibliothèque Qt-Embedded, le navigateur Opera et Embedded Linux de Red Hat.

Figure 2-7
Tablette Internet par Ericsson

La société française NETGEM (*http://www.netgem.com*) a développé une set top box (la NETBOX) utilisant initialement un système d'exploitation propriétaire. L'environnement graphique fut ensuite porté sur une version de Linux adaptée par NETGEM. La version actuelle utilise Hard Hat Linux de MontaVista (*http://www.mvista.com*). NETGEM diffuse également

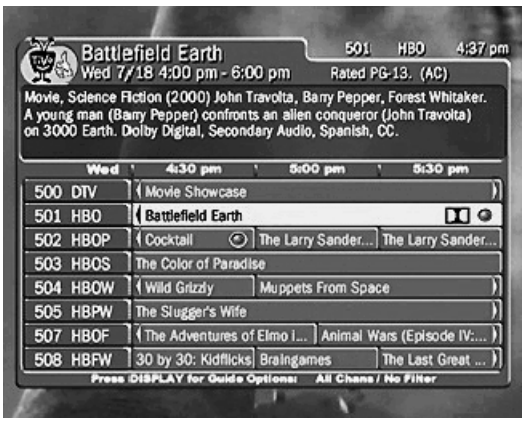
la suite logicielle NETGEM 4.0 basée sur Linux et destinée à l'intégration de fonctionnalités Internet dans les téléviseurs.

Les magnétoscopes numériques

La société américaine TiVo (*http://www.tivo.com*) a développé un système d'enregistrement numérique permettant l'enregistrement simultané de divers canaux de télévision. Le stockage s'effectue sur un disque dur optimisé pour les temps d'accès. L'interface de navigation, très conviviale, permet de programmer les enregistrements en fonction des goûts de l'utilisateur et de construire ainsi une télévision virtuelle basée sur les enregistrements.

Figure 2-8
Magnétoscope TiVo

Figure 2-9
Interface de TiVo



La société ReplayTV (*http://www.replaytv.com*) a développé un produit similaire.

Les routeurs

Le leader mondial des routeurs (CISCO Systems, *http://www.cisco.com*) utilise pour ses routeurs professionnels un système proche de la famille Unix, IOS. La gamme CISCO/Linksys est par contre basée sur Linux et les sources du système sont disponibles sur le site de la société (*http://www.linksys.com*). Ce produit est aujourd'hui un des leaders du marché des routeurs xDSL/Wi-Fi.

Le projet Open Source *Linux Router* (*http://www.linuxrouter.org*) a développé une distribution réduite permettant de construire un routeur efficace et à bon marché à partir d'un PC classique. Ce projet LRP (*Linux Router Project*) est parfois considéré comme une distribution embarquée à part entière.

Figure 2-10
Routeur CISCO/Linksys

Figure 2-11
Routeur xDSL de Lightning

La société suisse Lightning (*http://www.lightning.ch*) développe une gamme de routeurs utilisant Linux comme système embarqué.

De nombreux opérateurs Internet français comme Free SA, Neuf Télécom Cegetel ou Wanadoo fournissent des modems ADSL multi-fonctions (Freebox, Neuf box, C-Box, Livebox).

Ces « boîtes noires » permettent non seulement l'accès ADSL mais aussi l'accès local sans fil (Wi-Fi), la téléphonie sur IP (VoIP) ou bien l'accès à la télévision numérique (décodage MPEG2).

Figure 2-12
Freebox de Free SA.

La Freebox de Free SA est certainement le produit le plus répandu. Ses spécifications techniques avancées restent assez mystérieuses car Free communique peu sur ce point. Cependant l'utilisation de Linux dans ce terminal ne laisse aucun doute. Par défaut la Freebox se comporte en « proxy DHCP » et permet donc à l'utilisateur d'obtenir de manière transparente une adresse IP auprès du point d'accès Internet (DSLAM pour *Digital Subscriber Line Access Multiplexor*) de Free. Il est également possible de configurer la Freebox de manière à ce qu'elle se comporte comme un véritable routeur permettant de connecter à Internet plusieurs machines d'un réseau local privé masqué par la Freebox. La majeure partie des composants Linux de la Freebox est obtenue auprès du DSLAM lors de la mise sous tension ce qui fait que les diverses configurations sont effectuées au travers de l'espace de l'abonné sur le site de Free SA.

Outre les fonctions classiques d'accès à l'Internet haut-débit, la Freebox inclut des fonctionnalités intéressantes et évolutives. La description des services est disponible sur le site de Free à l'adresse *http://freebox.free.fr.*

Accès Wi-Fi

Il est possible d'ajouter une carte Wi-Fi PC-Card dans la Freebox (V3 et V4) afin de transformer la boîte en routeur Wi-Fi. La configuration s'effectue au travers de l'espace abonné sur le site de Free.

Téléphonie illimitée

La Freebox utilise la connexion ADSL pour permettre l'accès téléphonique en « voix sur IP » (VoIP). Pour cela, il suffit de connecter un combiné téléphonique classique sur la Freebox par un simple câble RJ11. Il est bien entendu possible de recevoir des appels sur un numéro spécial affecté par l'opérateur qui n'a rien à voir avec le numéro associé à la ligne téléphonique classique qui peut être gérée par un autre opérateur (cas de ligne non dégroupée).

Télévision et multimédia

On peut raccorder la Freebox à un téléviseur ou à une carte d'acquisition vidéo à la norme PAL. L'utilisation du service s'effectue grâce à une télécommande livrée avec la Freebox. Au niveau multimédia, Free met à disposition depuis peu le logiciel *Freeplayer* qui permet de diffuser sur un téléviseur des contenus multimédias provenant d'un ordinateur.

Quelques informations techniques

Voici quelques informations concernant la structure matérielle et logicielle de la Freebox. Ces informations proviennent de la société Free SA.

Le matériel

Historiquement, il existe 4 versions de la Freebox (V1 à V4). Les Freebox V1 et V2 utilisent un processeur MIPS (IDT RC32355), 16 Mo de mémoire vive et 4 Mo de mémoire flash. La gestion de la ligne DSL est assurée par un DSP. La télévision est gérée par un deuxième processeur spécifique (ST5518) utilisant un système d'exploitation propriétaire. La communication entre les deux processeurs est assurée par un FPGA. La partie téléphonie est également gérée par un DSP. Ces versions sont équipées d'une interface Ethernet 10/100 Mbps et d'une interface USB « device ».

La Freebox V3 est munie d'un processeur ARM sans MMU et de 8 Mo de mémoire vive. Dans ce cas, la gestion de la ligne DSL est intégrée au processeur. Cette version possède un afficheur en façade permettant de suivre l'état de démarrage du système (détection de ligne, connexion DSLAM, mise à jour logiciel, authentification). Elle inclut également un port PCMCIA destiné à recevoir l'extension Wi-Fi.

La version V4 est pourvue d'un processeur MIPS et de 16 Mo de mémoire vive. Cette version possède un port PCMCI/Cardbus ainsi qu'un port USB « host ». L'interface Ethernet est en 100 Mbps.

Le logiciel

La distribution Linux employée dans la Freebox a été entièrement créée par les équipes techniques de Free SA. Cette distribution utilise un système de génération inspiré par l'utilitaire *buildroot* (voir *http://buildroot.uclibc.org*). Cet utilitaire est constitué d'un ensemble de fichiers `Makefile` et `patch` permettant d'automatiser la génération d'une chaîne de compilation croisée et du système de fichiers principal.

La distribution utilise *uclibc* (*http://www.uclibc.org*) qui est une version plus réduite de la *libc* que la *glibc* habituellement mise en œuvre sous Linux. Le compilateur utilisé est le 3.3.5.

L'architecture est constituée d'un noyau Linux minimal résident en mémoire permanente (flash). À chaque démarrage de la Freebox, ce noyau télécharge un noyau plus complet auprès du DSLAM, ce qui permet d'assurer la présence du dernier logiciel « firmware » dans la Freebox.

La téléphonie

La société APLIO (*http://www.aplio.com*) a développé un équipement de téléphonie Internet basé sur une distribution Linux embarquée tournant sur un ARM7DTMI cadencé à 20 MHz. Ce système très réduit n'utilise pas d'interface graphique et occupe seulement 2 méga-octets de mémoire flash et 4 méga-octets de RAM.

Figure 2-13
Téléphone IP par APLIO

Indépendamment de cet équipement très particulier, certains constructeurs de centraux téléphoniques professionnels (PABX) utilisent d'ores et déjà des composants Linux.

Les caméras IP

Le principe d'une caméra IP est de transformer l'information analogique filmée par la caméra en un flux de données numériques exploitables par un système informatique connecté sur un réseau de type IP. Ces produits utilisent le plus souvent un format MJPEG (pour *Motion* JPEG) qui est constitué d'une suite d'images JPEG. La société suédoise AXIS (*http://www.axis.com*) a développé une caméra autonome comportant un système optique et un système d'exploitation Linux porté sur le processeur ETRAX, développé par AXIS. La caméra inclut un serveur HTTP qui permet un accès direct à partir de n'importe quel navigateur.

Figure 2.14
La caméra IP AXIS

Tableau 2-2 Récapitulatifs de quelques produits basés sur LINUX

Nom	Type	Fabricant	URL	Remarques
YOPI	PDA	SAMSUNG	http://www.yopi.com	Un des premiers PDA Linux, voir http://www.handhelds.org
Zaurus	PDA	SHARP	http://www.sharp.co.uk	Utilise Lineo et le navigateur Opera, voir http://www.handhelds.org
IPAQ	PDA	Compaq	http://www.compaq.com/products/iPAQ	Pas initialement sous Linux, voir http://www.handhelds.org
Bahia MPA	PDA	Silink	http://www.servelinux-fr.com/apropos/silink/	Toolkit disponible
Iplayer	Set top box	NetGem	http://www.netgem.com	
TiVo	Magnétoscope numérique	TiVo	http://www.tivo.com	Permet de construire un programme TV «virtuel»
ReplayTV	Magnétoscope numérique	ReplayTV	http://www.replaytvcom	Proche de TiVo
LinuxRouter	Routeur	Aucun	http://www.linuxrouter.org	Distribution pour construire un routeur à base de Linux réduit. Tient sur une disquette
MultiCom	Routeur	Lightning	http://www.lightning.ch	Routeur xDSL
Linksys	Routeur	CISCO/Linksys	http://www.linksys.com	Routeur xDSL/Wi-Fi, Sources Linux disponibles
Freebox	Routeur	Free SA	http://www.free.fr	Routeur/modem xDSL multi-fonction
Aplio	Téléphone IP	Aplio	http://www.aplio.com	
Axis 2100	Caméra IP	AXIS	http://www.axis.com	Intègre un serveur http

En résumé

Les systèmes embarqués propriétaires subissent certaines contraintes et ont en particulier quelques difficultés à suivre l'évolution technologique. Les logiciels Open Source ont de nombreux avantages, notamment en matière de disponibilité des sources et d'absence de royalties.

L'utilisation de l'Open Source pour les systèmes embarqués nécessite quelques précautions, surtout au niveau du support technique. De par des qualités reconnues, Linux est le plus souvent très bien adapté aux applications embarquées. Il sera cependant inadapté si l'empreinte mémoire disponible est trop réduite.

De nombreuses distributions Linux adaptées à l'embarqué existent déjà, soit sous forme de projets Open Source, soit sous forme de produits commerciaux (également Open Source le plus souvent).

De même, de nombreux produits industriels recourent à Linux dans des domaines aussi variés que les PDA, les set top boxes, les équipements multimédias, les routeurs ou la téléphonie.

3

Choix matériels pour un système Linux embarqué

Le chapitre précédent nous a permis de démontrer que Linux est un très bon choix pour un système embarqué. Le présent chapitre va être consacré à la description des choix matériels (hardware) les mieux adaptés à cet environnement.

Attention

Il va sans dire que les marques citées dans ce chapitre ne le sont pas à des fins publicitaires. Elles correspondent seulement à des produits utilisés par l'auteur pour des applications réelles.

Choix d'une architecture, PC ou non ?

Le monde de l'informatique est dominé par l'architecture dite *PC* héritée des systèmes personnels initialement développés par IBM au début des années 1980. En dépit des nombreux détracteurs et autres analystes qui prévoient sa fin proche, l'architecture PC a su s'adapter à l'évolution technologique en balayant sur son passage bon nombre de concurrents pourtant présentés comme révolutionnaires.

Dans le monde de l'ordinateur personnel, seul l'iMac d'Apple se taille une petite part de marché – bien inférieure à 10 % – au milieu de la déferlante des compatibles PC. Les raisons de ce succès durable du PC sont diverses :

- Le PC est depuis le début une architecture *ouverte*, ce qui a permis très rapidement le développement de machines compatibles à bas prix et donc d'imposer l'architecture sur un grand nombre de marchés, du très bas de gamme aux systèmes professionnels.

Par opposition, d'autres concurrents ont jalousement gardé leurs secrets et interdit la réalisation de clones, ce qui a limité la diffusion à des marchés plutôt haut de gamme. Ce qui est vrai pour l'architecture de base (la carte mère) l'est encore plus pour les périphériques comme les cartes d'extension, vendues souvent à des prix prohibitifs sur les architectures non-PC. Le fait est que bon nombre d'architectures concurrentes ont maintenant adopté des composants de l'architecture PC, comme le bus PCI *(Peripheral Component Interconnect)*, permettant l'utilisation de périphériques très compétitifs.

- Le PC a longtemps été exclusivement associé ou presque aux systèmes d'exploitation de Microsoft, la fulgurante ascension du PC découlant de ce qu'il ait été choisi pour développer le système d'exploitation DOS pour le premier IBM PC. Le fait est que les systèmes de Microsoft sont depuis longtemps les plus répandus et qu'ils existent presque exclusivement sur architecture PC.
- Enfin, le PC est depuis toujours associé au processeur Intel ou compatible. Intel s'est longtemps comporté en « Microsoft du processeur » agissant en quasi-hégémonie sur ce marché. Il est aujourd'hui talonné par d'autres fondeurs comme AMD, mais ces derniers se doivent de fournir des processeurs les plus compatibles possibles tout en restant plus performants que ceux d'Intel.

Le problème est différent dans le monde de l'embarqué pour lequel l'architecture Intel/PC n'a jamais eu – ou n'a pas encore – la position hégémonique qu'elle connaît dans le monde de l'informatique classique. Une des raisons tient au processeur car Intel s'est fortement concentré sur le marché du PC classique et ses produits ne remplissent pas toujours les contraintes de performance, consommation et dissipation thermique nécessaires à certains environnements embarqués. En ce qui concerne des architectures de même niveau de complexité, le processeur *PowerPC* codéveloppé par IBM et Motorola et utilisé dans la gamme Macintosh d'Apple est également très utilisé dans le monde industriel en raison de son très bon rapport consommation/performances.

L'autre raison a trait à l'architecture elle-même. Le principal intérêt du PC est le côté interchangeable et banalisé de ses composants. Le développeur ou la secrétaire qui utilise sa machine dans un environnement stable et sécurisé (pas de contrainte d'humidité, de vibration ou de température) pourra aisément remplacer une carte mère par une autre, équivalente, et ce à très bon prix. Le plus souvent, les systèmes sont en fait peu sollicités car utilisés seulement quelques heures par jour. Les systèmes plus sollicités comme les serveurs sont placés dans des conditions de fonctionnement encore plus idéales, souvent dans des salles climatisées, et choyés par des administrateurs système soucieux de leur bien-être.

Il en va autrement pour nombre de systèmes embarqués, contraints de fonctionner 24 heures sur 24 dans des environnements beaucoup plus agressifs. Ce dernier point peut conduire à l'utilisation de composants de meilleures qualités (on parle de « PC industriel »), donc bien plus onéreux. Le mythe du PC à bon marché, banalisé et multi-usage que l'on peut acheter au revendeur du coin ou pour trois fois rien à Taïwan en prend un coup, même si ça déçoit profondément votre patron ou votre responsable des achats…

Faut-il donc délaisser l'architecture PC ? Certainement pas. Quelle que soit l'issue du développement d'un projet de système embarqué, il est un domaine sur lequel le PC reste imbattable, celui du maquettage et de la simulation. Même si le système final utilise un processeur PowerPC ou StrongARM sur une carte étudiée par vos soins, il y a gros à parier que les premières versions seront développées sur architecture PC. De même, sur de petites séries ou des systèmes fonctionnant dans des conditions extérieures raisonnables, le choix d'une architecture PC reste économique et très viable d'un point de vue technique.

En résumé, les règles d'or à respecter sont les suivantes :

- Sauf cas exceptionnel, utilisez une architecture PC pour la phase de maquettage et d'étude de fonctionnalité. Il est toujours plus facile et souvent moins onéreux de montrer une maquette tournant dans un PC portable sous Linux que sur un prototype qui refusera de fonctionner à plus d'un kilomètre de votre bureau d'étude.
- N'essayez pas d'adapter votre projet à l'architecture PC, ou faites-le dans des limites très raisonnables.
- Linux est certainement le système le plus disponible sur nombre de processeurs, en tout cas pour la plupart de ceux utilisés dans des environnements embarqués. De plus, le portage du noyau Linux vers un nouveau processeur est un travail relativement aisé de par la littérature conséquente et les nombreux travaux réalisés dans ce domaine. Ce point est également vrai pour la chaîne de développement GNU (gcc et gdb).
- Limitez une étude matérielle à une production importante ou bien prenez de sérieuses garanties sur le prix de vente.

Choix du processeur : MMU ou non ?

Le concept du MMU

Linux a été initialement développé sur la base du mécanisme de *mémoire protégée* du processeur Intel 80386. Ce mécanisme, qui repose sur un composant matériel appelé MMU (*Memory Management Unit*), permet à un processus de ne jamais écraser l'espace mémoire d'un autre processus. La MMU autorise la conversion entre les adresses physiques – adresses effectivement utilisées dans la machine – et les adresses logiques – adresses vues par le processus et allouées par le système d'exploitation. Si un processus tente de sortir par erreur de l'espace mémoire qui lui est accordé, la MMU détecte l'erreur et stoppe le programme en générant une erreur de « violation de segmentation » (*segmentation violation ou SIGSEGV*). De ce fait, un programme tournant sous Linux dans l'espace dit « utilisateur » – par opposition à l'espace « noyau » – ne peut jamais « planter » le système.

Les versions courantes du noyau Linux sont prévues pour fonctionner sur des processeurs avec MMU, ce qui concerne la majorité des processeurs utilisés dans la micro-informatique classique et aussi dans un bon nombre d'applications embarquées. En revanche, ces processeurs sont en général plus gourmands en ressources matérielles, et

certaines applications dites « profondément enfouies » (*deeply embedded*) ne pourront utiliser que des processeurs dépourvus de MMU.

Il existe bien entendu des systèmes d'exploitation dédiés à ces micro-contrôleurs – µC/OS en est un très bon exemple – mais ceux-ci sont en général bien plus limités que Linux au niveau des protocoles standards et de l'interopérabilité avec le monde extérieur.

µClinux: Linux sans MMU

Un portage du noyau Linux est cependant disponible pour les processeurs dépourvus de MMU : µClinux – pour Micro-C Linux (Linux pour micro-contrôleurs) mais à prononcer *You see Linux* (*http://www.uclinux.org*). Le portage est initialement basé sur la version 2.0.38 du noyau Linux mais des versions basées sur les noyaux 2.4 et 2.6 sont également disponibles.

L'API du système est identique au vrai noyau Linux bien que µClinux utilise une libC – bibliothèque de base de programmation – différente de la glibc de la version standard de Linux. La motivation est toujours la même : le gain d'espace, lorsqu'on sait que les versions récentes de la glibc ont une taille largement supérieure au méga-octet. La principale limitation de µClinux par rapport à Linux est l'absence de protection de mémoire entre les processus mais également avec le noyau. De ce fait, une application erronée pourra facilement « planter » le système. L'utilisation du système µClinux sera détaillée dans la deuxième partie de cet ouvrage, au chapitre 10.

Les processeurs compatibles x86

Concernant les processeurs avec MMU, et outre les classiques Pentium et autres Celeron fournis par Intel, de nombreux fondeurs comme AMD (*http://www.amd.com*), National Semiconductor (*http://www.national.com*) ou VIA (*http://www.viavpsd.com*) proposent des processeurs compatibles x86 souvent d'un excellent rapport qualité/prix.

Le processeur ELAN520 développé par AMD est déjà très implanté dans le monde des applications embarquées à faible coût. Il est annoncé comme officiellement compatible LINUX sur le site AMD (*http://www.amd.com/epd/linux*).

Le processeur Geode développé par National équipe un grand nombre de cartes mère industrielles dont les cartes PC/104 citées plus bas.

Le nouveau processeur VIA C3 cadencé à 1 GHz peut également être un choix très intéressant (*http://www.via.com.tw/en/viac3/c3.jsp*). Il est d'ores et déjà intégré sur des cartes mère à très faible coût. Le site *http://www.viaarena.com* regroupe déjà un grand nombre d'informations sur ce processeur. Une section dédiée à Linux est également disponible sur le site.

Bien que moins puissant (il est détecté comme compatible 486 par le noyau Linux), le processeur STPC (*http://www.st.com/stpc*) est également un choix intéressant pour des applications ne demandant pas de fortes puissances de calcul.

La société Transmeta (employeur de Linus Torvalds) commercialise un processeur à faible consommation, compatible x86, et orienté embarqué, appelé Crusoe. Une distribution Linux est disponible pour cette architecture (Midori Linux), voir *http://www.transmeta.com/developers*.

Les autres processeurs

L'architecture x86 n'est pas forcément la meilleure pour toutes les applications, loin s'en faut. Dans le cas de la production d'une série de cartes mère assez importante (plusieurs milliers d'exemplaires) ou d'un besoin particulier (faible consommation, intégration), d'autres processeurs sont plus avantageux. Nous pouvons citer l'architecture PowerPC (PPC) largement utilisée dans les applications industrielles.

Concernant des applications tournant sur des cartes mère de taille réduite, l'architecture ARM est de plus en plus utilisée sous diverses formes (ARM7, ARM9, XSCALE, etc.). Ces processeurs ont un bon niveau d'intégration, une faible consommation et permettent donc la conception de cartes mère plus simples et moins coûteuses que les x86. Le processeur AT91RM9200 produit par la société ATMEL (*http://www.atmel.com*) en est un bon exemple car il intègre en standard un contrôleur Ethernet 10/100 Mbits/s, une interface USB host et une interface USB device.

La mémoire de masse

La mémoire de masse est utilisée pour stocker l'image du système d'exploitation et les données lues ou écrites au cours du fonctionnement de ce système. Les systèmes de micro-informatique classiques utilisent des disques durs compatibles avec les standards de bus IDE (*Integrated Drive Electronics*) ou SCSI (*Small Computer System Interface*).

La problématique des systèmes embarqués est différente car ils utilisent en général un faible espace de stockage, du moins pour le système d'exploitation. La ou les distributions classiques installent plus d'un giga-octet de programmes sur des disques dont le moins volumineux accepte jusqu'à 10 Go, les distributions Linux embarquées se situent entre 2 et 10 Mo et les périphériques de stockage sont de ce fait très différents.

D'autres facteurs peuvent également être déterminants :

- la fragilité des disques durs lorsqu'ils sont utilisés dans des environnements mobiles ;
- la consommation d'énergie et la dissipation thermique ;
- le bruit généré par les disques classiques.

Les systèmes embarqués utilisent dans la plupart des cas des mémoires permanentes appelées « mémoires flash » (*flash memory*). La facilité d'utilisation des flash est similaire à celle de la mémoire vive bien que son coût soit bien supérieur. La mémoire flash peut se présenter sous différentes formes :

- Des circuits ou *chips* classiques directement soudés sur la carte mère ou des circuits amovibles comme les DiskOnChip de chez M-Systems (*http://www.m-sys.com*). Dans ce cas, le noyau Linux doit contenir un pilote de périphérique – ou driver – dédié. La photo ci-après montre la mémoire flash DOC2000 de M-Systems existant actuellement pour des capacités allant de 16 à 568 Mo. M-Systems supporte officiellement le noyau Linux et fournit en téléchargement les composants logiciels (patch) nécessaires à l'ajout du pilote aux noyaux 2.0, 2.2 et 2.4. Les noyaux Linux 2.4 et 2.6 contiennent également le pilote MTD (*Memory Technology Devices*) utilisable sur un grand nombre de mémoires flash dont les DiskOnChip.

Figure 3-1
DOC2000 de M-Systems.

- Des circuits intégrés dans des boîtiers standards comme les *CompactFlash* ou des boîtiers compatibles avec la géométrie des disques durs 2,5 pouces utilisés dans les ordinateurs portables. Dans ce cas, les disques flash sont le plus souvent accessibles en utilisant une interface IDE standard, ce qui évite l'ajout d'un pilote et facilite l'intégration sur des PC classiques lors des phases de développement.

Figure 3-2
CompactFlash avec interface IDE.

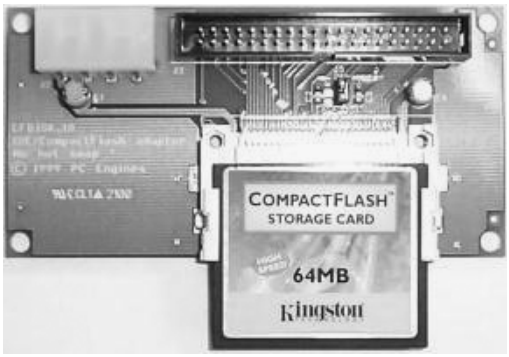


Figure 3-3
Disque Flash IDE 2,5 pouces.

Outre le prix, les mémoires flash ont cependant quelques limitations par rapport à la mémoire vive :

- le temps d'accès, en particulier pour la phase d'écriture, est supérieur à celui de la mémoire vive ;
- la durée de vie de la « flash » est fortement influencée par le nombre d'écritures par unité de temps. Le site de M-Systems fournit une page permettant d'évaluer la durée de vie de ses produits en fonction du type, de la capacité et la fréquence d'écriture.

Les bus d'extension et de communication

Les bus d'extension ISA et PCI

L'architecture Intel (entre autres) fournit différents bus d'extension et de communication. L'ancien bus ISA (*Industry Standard Architecture*) est souvent présent sur les cartes mère dites « industrielles » à cause de la nécessaire compatibilité avec d'anciennes cartes d'extension faiblement diffusées. Ce bus a cependant tendance à disparaître et il n'est plus disponible sur les cartes mère grand public à plus bas coût. Dans la mesure du possible, il est préférable d'utiliser un bus PCI qui est beaucoup plus facile à manipuler sous Linux car bien plus récent et donc effectuant automatiquement la plupart des initialisations matérielles, comme les niveaux d'interruption. Si cela est nécessaire, le développement d'un pilote de carte PCI sous Linux est relativement aisé (plus qu'en ISA) car l'API du noyau propose un grand nombre de fonctions de haut niveau. Pour des informations plus complètes concernant le développement de tels pilotes, nous vous renvoyons à l'excellent ouvrage d'Alessandro Rubini, *Linux Device Drivers*, disponible en ligne à l'adresse *http://www.xml.com/ldd/chapter/book* et *http://www.oreilly.com/catalog/linuxdrive2*.

Ajoutons à cela que le débit disponible sur le bus PCI est beaucoup plus important que celui du bus ISA (jusqu'à 1 Go/s contre 8 Mo/s). Certaines cartes miniatures proposent un bus au format dit CompactPCI qui propose les mêmes caractéristiques que le bus PCI avec un encombrement mécanique plus faible, même si le coût des cartes est plus élevé. Le débit est également plus faible que celui du PCI classique puisqu'il est limité à 132 Mo/s.

Les ports séries

Les ports séries RS-232 sont traditionnellement très utilisés dans les systèmes embarqués de par leur ancienneté et leur facilité de mise en œuvre. Pour la même raison que le bus ISA, ils ont cependant tendance à disparaître du matériel grand public au profit de bus plus modernes comme l'USB. La plupart des cartes mère proposent cependant au moins un port série et il est aisé d'ajouter des cartes d'extensions au format ISA ou PCI. Ces cartes peuvent être équipées de contrôleurs classiques de type 16550A, et le noyau Linux pourra les piloter sans aucune modification à partir du moment où le support du port série standard est activé dans la configuration du noyau. La détection des ports peut être vérifiée au moyen de la commande suivante :

```
# dmesg | grep tty
ttyS00 at 0x03f8 (irq = 4) is a 16550A
```

La commande Linux **setserial** permet également de manipuler les ports séries détectés.

```
# setserial /dev/ttyS0
/dev/ttyS0, UART: 16550A, Port: 0x03f8, IRQ: 4
```

Le noyau Linux inclut également le support d'un grand nombre de cartes séries dites « multi-voies » et permettant d'ajouter un plus grand nombre de ports séries sur une seule carte d'extension (de 4 à 16). Dans ce cas-là, les options adéquates doivent être validées dans la configuration du noyau (**make xconfig**) dans la rubrique *Character devices*.

Le bus USB

Le bus USB (*Universal Standard Bus*), récemment introduit, présente un certain nombre d'avantages tant au niveau de la possibilité de connecter des périphériques « à chaud » que du débit disponible (environ 1,2 Mo/s). Il est très répandu dans les systèmes grand public depuis Windows 98 sur PC ou l'iMac. Le support de l'USB sur Linux n'est officiel que depuis la version 2.4 et tous les périphériques ne sont pas supportés (loin de là) car cela nécessite le développement d'un pilote et donc de disposer des spécifications matérielles du périphérique. Dans tous les cas, même si les spécifications sont alléchantes, il est assez peu recommandé dans des applications sensibles car sa stabilité n'est pas encore au niveau du support PCI, ISA ou Ethernet. La liste des périphériques supportés est cependant disponible sur le site *http://www.linux-usb.org*.

Les autres bus : I2C, I2O, IEEE

De nombreux autres supports bus et protocoles associés sont disponibles d'ores et déjà dans l'arborescence officielle des sources du noyau Linux. Même si le support de certains bus est éprouvé (I2C), la réalisation d'un projet intégrant un bus de ce type devra passer par la validation de l'état du support en consultant la page web associée au projet. Les pointeurs sont en général disponibles dans le répertoire **Documentation** des sources du noyau Linux.

Nous pouvons citer ici quelques pointeurs intéressants concernant le support de divers formats de bus sur Linux.

- La page du Linux Lab Project sur *http://www.llp.fu-berlin.de*. Ce projet concerne le développement de divers bus industriels sur Linux.
- La page du bus IEEE1394 appelé aussi FireWire. Est située sur *http://linux1394.sourceforge.net*. Le support est disponible dans le noyau 2.4. Le débit de ce bus de plus en plus utilisé est d'environ 50 Mo/s.

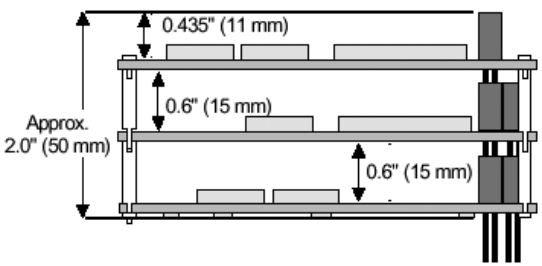
Les cartes PC/104

Le format PC/104 est apparu en 1992 ; c'est en fait une version compacte du format PC classique. Le principe du PC/104 est de fournir des modules de petite taille que l'on peut empiler les uns sur les autres. L'encombrement d'une carte PC/104 est comparable à celui d'une disquette 3,5 pouces (90 sur 96 mm), ce qui permet de construire des systèmes très peu volumineux.

Le PC/104 utilise initialement le format de bus ISA ; cependant, il existe une évolution du format, appelée PC/104-*Plus*, qui permet de disposer en plus des connecteurs ISA d'un connecteur PCI. Le nom du format a pour origine la somme du nombre de broches disponibles sur le bus ISA de la carte PC/104, soit 40 plus 64.

Le schéma ci-après donne un exemple d'empilage de trois cartes PC/104.

Figure 3-4
Configuration PC/104 à 3 cartes.



Le schéma ci-après indique la géométrie d'une carte PC/104-Plus avec connecteur PCI (J3), ainsi que les deux connecteurs ISA du format PC/104 classique (J1 et J2).

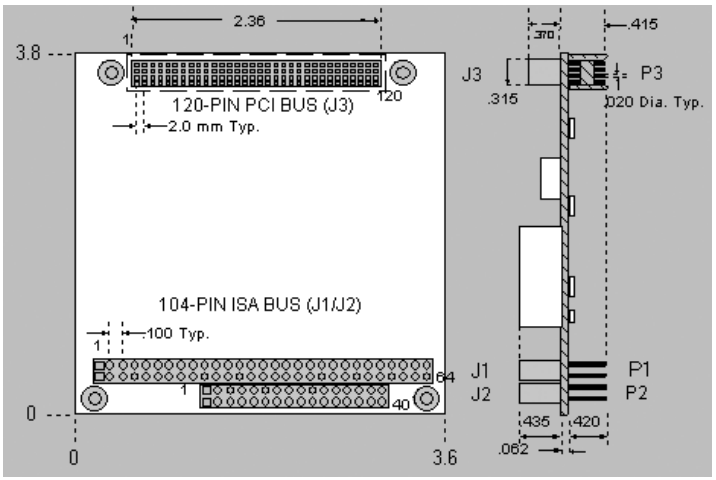
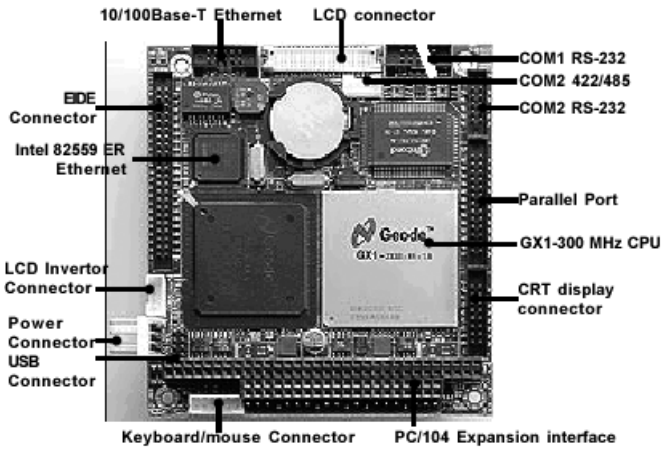


Figure 3-5
Carte PC/104-Plus.

Le principal avantage du format PC/104 est sa forte similitude avec l'architecture PC standard. Des cartes mère intégrées disposeront de toutes les interfaces classiques (disque dur IDE 2,5 pouces, CompactFlash IDE, lecteur de disquette, contrôleur Ethernet, USB, port série et parallèle, connecteurs VGA, clavier et souris PS/2), ce qui permettra de migrer très facilement d'un environnement de développement vers l'environnement embarqué définitif. De par sa forte intégration, ce type de carte est cependant relativement onéreux.

La société Advantech (*http://www.advantech.com*) est leader sur le marché des cartes processeurs PC/104. La carte PCM-3350 décrite ci-après est basée sur un compatible processeur NS Geode à 300 MHz (compatible x86 Intel) et dispose de toutes les interfaces d'un PC classique ainsi que d'un connecteur CompactFlash vu comme un disque IDE. La carte dispose également d'un connecteur mémoire DIMM (*Dual In-line Memory Module*), standard permettant d'utiliser jusqu'à 128 Mo de RAM.

Figure 3-6
Carte Advantech PCM-3350.



La page du consortium PC/104 contenant la liste des fabricants est disponible sur *http://www.pc104.org*.

Les cartes DIL

Ces cartes permettent un très haut niveau d'intégration car elles intègrent un système complet sur un format DIL (*Dual In Line*) à 64 broches, comme cela est indiqué sur le schéma ci-contre.

La carte existe en version compatible Intel 486 et StrongARM. Elle est très bien adaptée à des applications réseau car elle contient un contrôleur Ethernet et peut gérer jusqu'à trois ports séries. La tension d'alimentation n'est que de 5 V. La carte est équipée de 2 Mo de flash et 8 Mo de RAM.

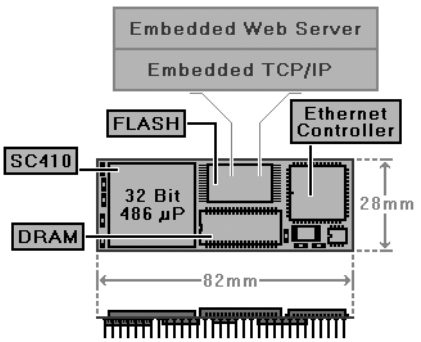


Figure 3-7 *Carte DIL.*

La société SSV-Embedded (*http://www.ssv-embedded.de*) propose un kit d'évaluation contenant une version de Linux (noyau 2.2) adaptée à ce type de carte. La configuration

standard est basée sur une distribution DOS très légère qui permet de démarrer Linux depuis DOS grâce à l'utilitaire LOADLIN. L'utilisation de LOADLIN sera détaillée dans la deuxième partie de l'ouvrage, au chapitre 8.

Des informations sur ce produit sont disponibles sur *http://www.dilnetpc.com*. Le site est extrêmement bien documenté et contient un grand nombre de documents techniques à télécharger au format PDF.

Les cartes uCsimm

Le format uCsimm est un format micro-contrôleur SIMM (*Single In-line Memory Module*) à 30 broches développé autour de µClinux, portage du noyau Linux pour les processeurs dépourvus de MMU. Le module utilise un processeur Motorola DragonBall 68EZ328 et dispose de 2 Mo de flash et 8 Mo de RAM.

Des kits d'évaluation et de développement sont disponibles sur *http://www.uclinux.com/orderdesk*.

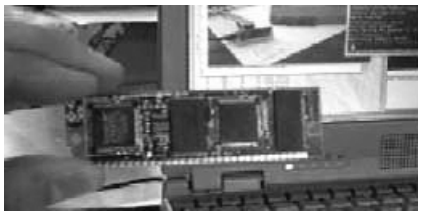


Figure 3-8
Module uCsimm.

Tableau 3-1. Récapitulatifs de quelques composants matériels

Nom	Type	Fabricant	URL	Remarques
Geode	Processeur compatible x86	National Semiconductor	*http://www.national.com*	Dérivé du Cyrix MediaGX
VIA C3	Processeur compatible x86	VIA	*http://www.via.com.tw/en/viac3/c3.jsp*	Excellent rapport performance/consommation. Dérivé du MediaGX
Elan 520	Processeur compatible x86	AMD	*http://www.amd.com/epd/linux*	Très répandu
STPC	Processeur compatible x86	ST	*http://www.st.com/stpc*	
ARM AT91RM9 200	Processeur famille ARM9	ATMEL	*http://www.atmel.com*	Bon processeur industriel. Intègre un contrôleur Ethernet, un contrôleur USB host et device
Mini-ITX	Carte mère	EPIA/VIA	*http://www.viavpsd.com*	Basée sur le VIA C3, très bon rapport performance/prix
Disk On Chip	Mémoire flash	M-Systems	*http://www.m-sys.com*	Support dans le noyau 2.4 (sauf Millenium Plus) ou chez M-Systems (patch noyau). Attention au problème de GPL dans le cas du patch M-Systems
Compact-Flash	Mémoire flash	Plusieurs	*http://www.compact-flash.org*	En général avec interface IDE, donc pas de pilote à ajouter
PC/104	Cartes mère	Plusieurs	*http://www.pc104.org*	Faible encombrement mais coûteux
DIL	Cartes mère	SSV	*http://www.dilnetpc.com*	
µCsimm	Carte mère	Lineo	*http://www.uclinux.com*	Pour µClinux, très faible encombrement

En résumé

L'architecture PC est aujourd'hui un choix intéressant pour un système Linux embarqué, en particulier pour la phase de maquettage. D'autres choix comme le PowerPC sont également très viables. Une grande série sur un système plus enfoui peut justifier l'utilisation d'architectures telles que le StrongARM.

Le portage µClinux peut fonctionner sur des processeurs sans MMU (en général, des micro-contrôleurs). Ces processeurs génèrent des coûts de mise en œuvre inférieurs mais induisent des contraintes de fonctionnement (dues à la protection de la mémoire).

Il existe aujourd'hui de nombreux périphériques « disque flash », compatibles Linux.

Les formats PC/104, cartes DIL et uCsimm sont répandus dans le monde Linux embarqué et l'on trouve des kits de développement qui facilitent leur intégration.

Deuxième partie

Méthodologie de création d'un système Linux embarqué

Après une présentation détaillée des principaux éléments du noyau Linux, nous décrirons une méthode permettant de réaliser une version embarquée de Linux à partir de composants d'une distribution classique.

Nous traiterons ensuite la mise en place et l'optimisation de composants comme la couche réseau ou bien les différentes interfaces graphiques disponibles dans un environnement réduit. Les différentes méthodes de débogage adaptées à l'environnement embarqué seront également détaillées. Enfin, nous aborderons quelques techniques particulières mais essentielles comme la gestion des mémoires « flash » et des systèmes de démarrage.

Plusieurs chapitres seront consacrés à des versions spéciales de Linux comme les extensions temps-réel (RTLinux et RTAI) et la version pour micro-contrôleur µClinux. Et pour finir, un chapitre sera dédié à l'utilisation de l'environnement Open Source PeeWeeLinux.

4

Structure de Linux

Ce chapitre décrit les éléments principaux de la structure de Linux.

Les tests et manipulations décrits dans ce chapitre pourront être réalisés sur un système installé avec une distribution classique. À quelques exceptions près, la structure du système est calquée sur les autres systèmes Unix libres ou propriétaires à savoir :

- un noyau ou kernel réalisant les fonctions essentielles comme la gestion des tâches et de la mémoire, ainsi que l'interfaçage entre le matériel et les applicatifs grâce entre autres aux pilotes de périphériques ou en anglais *device drivers* ;
- une bibliothèque principale appelée libc ou glibc (GNU libc) sous Linux et contenant les fonctions de base utilisées par les applicatifs ;
- les applicatifs eux-mêmes, appelés également commandes, soit livrés en standard avec le système ou bien développés pour des besoins spécifiques.

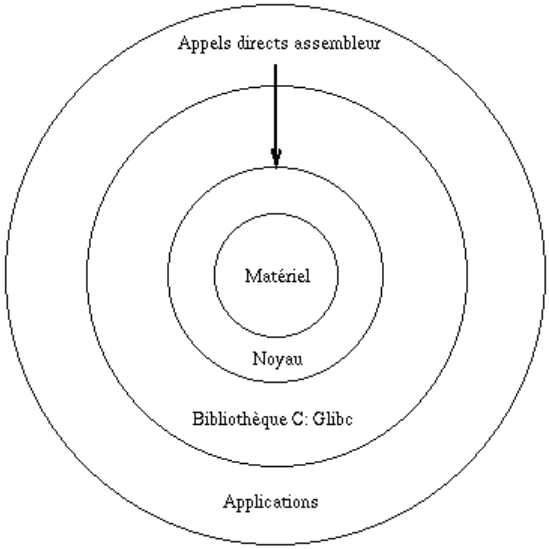
Remarques

Cet ouvrage étant destiné spécifiquement à l'utilisation de Linux dans des applications embarquées, notre intention n'est pas de nous livrer à une étude approfondie des différents composants du système mais plutôt de présenter les éléments indispensables à la réalisation d'un système embarqué.

Les exemples présentés dans ce chapitre sont basés sur le noyau 2.4 mais les concepts sont également vrais pour le noyau 2.6. En cas de différence notable entre les deux versions, celle-ci sera explicitée.

La structure du système est de forme concentrique comme décrit dans la figure 4-1 ci-après.

Figure 4-1
Structure du système.



À ces éléments standards il faut ajouter un programme de démarrage ou *bootstrap* comme LILO (*LInux LOader*). À la différence des autres composants du système, le programme de démarrage ou chargeur est très dépendant du matériel utilisé. Quelques éléments de la configuration de LILO sont explicités dans ce chapitre.

Le schéma de démarrage d'un système Linux est relativement simple et on peut le décomposer comme suit :

1. Chargement du système par LILO ou un programme équivalent, comme GRUB.
2. Chargement du noyau Linux. Le chargement s'accompagne de l'initialisation des périphériques matériels indispensables au démarrage, et donc au chargement des pilotes de périphériques associés. Le noyau Linux tente également de charger sa partition principale ou *root partition* sur laquelle il ira chercher les éléments nécessaires à la suite du lancement du système.
3. Lorsque le noyau Linux est chargé, il exécute un programme d'initialisation qui, par défaut, correspond à la commande **`/sbin/init`**. Cependant, on peut indiquer facilement au noyau de passer la main à un autre programme.
4. Dans le cas de l'utilisation du programme **`init`** standard, ce dernier explore le fichier de configuration **`/etc/inittab`** qui contient le chemin d'accès à un script de démarrage, comme ceci :

```
# System initialization (runs when system boots).
si:S:sysinit:/etc/rc.d/rc.d/rc.sysinit
```

Le script en question poursuit l'initialisation du système.

Le noyau Linux

Le noyau est l'élément principal du système, et ce pour plusieurs raisons. La première raison est historique puisque ce noyau fut initialement conçu par Linus Torvalds, créateur du système Linux, le reste du système étant constitué de composants provenant en majorité du projet GNU de Richard Stallman. L'autre raison est technique et tient en l'occurrence à la structure monolithique du noyau qui en fait l'interface unique entre le système et le matériel dans la quasi-totalité des cas de figure.

Structure globale du noyau

Dans le cas de Linux, le noyau est un fichier exécutable, monolithique ou presque, chargé d'assurer les fonctions essentielles du système comme la gestion des tâches, ou *scheduling*, la gestion de la mémoire et le pilotage des périphériques que ceux-ci soient réellement des périphériques matériels ou bien qu'ils soient des périphériques virtuels comme les systèmes de fichiers que nous décrirons plus en détail dans la suite de l'ouvrage.

Dans une distribution Linux classique, le noyau est physiquement représenté par un fichier localisé sur le répertoire **`/boot`** :

```
# ls -l /boot/vmlinuz-2.4.7-10
-rw-r--r--    1 root     root       802068 sep  6  2001 /boot/vmlinuz-2.4.7-10
```

Le nom du noyau est libre mais il est généralement suffixé en fonction de la version du noyau, ici 2.4.7 et de l'*extra-version* choisie par celui qui a généré ce noyau, soit 10 dans le cas présent. On peut avoir de même :

```
# ls -l /boot/bzImage*
-rw-r--r--    1 root     root       952696 fév  7 14:27 /boot/bzImage-2.4.14
-rw-r--r--    1 root     root       761341 mai 28 16:51 /boot/bzImage-2.4.16_plockb
-rw-r--r--    1 root     root       767760 mai 27 17:06 /boot/bzImage-2.4.16_rtsched
```

Nous verrons plus tard que l'extra-version est définie dans le fichier **`Makefile`** de l'arborescence des sources du noyau.

Le noyau Linux utilise également un fichier nommé **`System.map`** qui contient des informations sur des adresses internes du noyau. Ces informations sont utiles à la gestion des modules décrits ci-après. Le fichier **`System.map`** est également présent sur le répertoire **`/boot`**. En toute rigueur, il est lié à la version complète du noyau généré, y compris l'extra-version, car il est généré lors de la compilation du noyau.

Les modules chargeables du noyau

Dans les cas d'applications classiques de Linux, le noyau utilise le plus souvent des modules qui peuvent être dynamiquement chargés et déchargés en fonction des besoins du système. Ces modules peuvent être des pilotes de périphériques matériels, comme des cartes d'extension, ou bien liés à un support générique de plus haut niveau comme le support SCSI. L'utilisation des modules permet d'optimiser la mémoire du système à un instant donné car un pilote non utilisé pourra être déchargé, libérant ainsi sa mémoire.

De même, l'utilisation des modules permettra d'ajouter dynamiquement des périphériques sans redémarrer le système.

Dans le cas d'un noyau embarqué, on pourra cependant se poser la question quant à l'utilisation des modules sachant que ceux-ci nécessitent la validation d'options spécifiques lors de la compilation du noyau, ce qui augmente légèrement sa taille. De même, la hiérarchie des modules nécessite la mise en place du répertoire `/lib/modules`, ce qui augmente le nombre de fichiers, la complexité du système et aussi légèrement l'espace occupé par ce dernier.

Le choix de l'utilisation ou non des modules sera laissé à la charge de l'intégrateur du système en fonction de ses besoins. Différents cas de figure seront envisagés dans les chapitres concernant la réduction du système.

```
# ls -l /lib/modules
total 52
drwxr-xr-x    4 root     root         4096 mai 27 16:45 2.4.18-rtsched
drwxr-xr-x    5 root     root         4096 mai 23 11:44 2.4.4-rtl
drwxr-xr-x    4 root     root         4096 fév 13 17:08 2.4.7-10
drwxr-xr-x    4 root     root         4096 nov 25  2001 2.4.9-13
drwxr-xr-x    4 root     root         4096 fév  7 14:05 2.4.9-21
...
```

À chaque sous-répertoire correspond une version du noyau. Dans le cas présent, les modules utilisés par le noyau seront localisés dans le répertoire `2.4.7-10`.

```
# ls -l /lib/modules/2.4.7-10/
total 228
lrwxrwxrwx    1 root     root           31 nov  1  2001 build -> ../../../usr/src/
                                                          ➥ linux-2.4.7-10
drwxr-xr-x    7 root     root         4096 nov  1  2001 kernel
-rw-r--r--    1 root     root        82111 mar 25 17:40 modules.dep
-rw-r--r--    1 root     root           31 mar 25 17:40 modules.generic_string
-rw-r--r--    1 root     root           73 mar 25 17:40 modules.ieee1394map
-rw-r--r--    1 root     root         7673 mar 25 17:40 modules.isapnpmap
-rw-r--r--    1 root     root           29 mar 25 17:40 modules.parportmap
-rw-r--r--    1 root     root        45571 mar 25 17:40 modules.pcimap
-rw-r--r--    1 root     root        59973 mar 25 17:40 modules.usbmap
drwxr-xr-x    2 root     root         4096 nov  1  2001 pcmcia
```

Les modules sont répartis dans des sous-répertoires selon une classification fonctionnelle. La liste ci-après permet de visualiser les modules correspondant à des pilotes réseau :

```
# ls -l /lib/modules/2.4.7-10/kernel/drivers/net | head
total 2348
-rw-r--r--    1 root     root         9852 sep  6  2001 3c501.o
-rw-r--r--    1 root     root         8912 sep  6  2001 3c503.o
-rw-r--r--    1 root     root        24996 sep  6  2001 3c505.o
-rw-r--r--    1 root     root        10304 sep  6  2001 3c507.o
-rw-r--r--    1 root     root        13652 sep  6  2001 3c509.o
-rw-r--r--    1 root     root        22568 sep  6  2001 3c515.o
-rw-r--r--    1 root     root        41920 sep  6  2001 3c59x.o
```

```
-rw-r--r--    1 root     root        22072 sep  6  2001 8139too.o
-rw-r--r--    1 root     root        22160 sep  6  2001 82596.o
```

Le fichier **modules.dep** contient les dépendances entre les modules sous la forme d'une simple liste contenant une définition de dépendance par ligne. Cette liste est générée au démarrage du système par la commande **depmod -*a***.

On doit également utiliser cette commande chaque fois que l'on ajoute un nouveau module à l'arborescence des modules. Un extrait de la liste est présenté ci-après :

```
/lib/modules/2.4.7-10/kernel/fs/vfat/vfat.o:
/lib/modules/2.4.7-10/kernel/fs/fat/fat.o
```

La ligne ci-après montre que le module **vfat.o** nécessite la présence du module **fat.o**.

Bien que les modules soient normalement chargés de manière automatique par le système, nous allons décrire en quelques lignes les principales commandes de manipulation des modules, et ce afin de donner un bon aperçu des opérations possibles sur les modules.

Remarque

Les modules sont manipulés grâce à un paquetage nommé modutils. En cas d'utilisation du noyau 2.6, il est nécessaire d'utiliser une version récente du paquetage (module-init-tools-3.0 ou supérieur) disponible en téléchargement avec les sources du noyau. Ces dernières versions sont compatibles 2.4 et 2.6.

On peut forcer le chargement d'un module en utilisant la commande **insmod**. Prenons l'exemple d'un module **hello.o** sans dépendance avec aucun autre module. On peut charger ce module au moyen de la commande :

```
# insmod hello.o
```

On peut aussi ajouter ce module à l'arborescence des modules standards en effectuant :

```
# cp hello.o
/lib/modules/2.4.7-10/kernel/drivers/char/lib/modules/2.4.7-10/kernel/driver/char
# depmod -a
```

puis charger ce module avec la commande :

```
# insmod hello
Using /lib/modules/2.4.7-10/kernel/drivers/char/hello.o
```

Notez qu'il est nécessaire d'être super-utilisateur pour charger un module, et, également, que dans le cas où le module est installé dans l'arborescence, on ne spécifie pas le suffixe. La trace du chargement effectif du module est visible dans le fichier des messages du noyau. On peut également vérifier sa présence en utilisant la commande **lsmod** :

```
# dmesg | tail -1
Hello world!
# lsmod
Module                  Size  Used by
hello                    292   0  (unused)
```

Lors du chargement du module, il est possible de spécifier des paramètres par la ligne de commande :

```
insmod bttv card=39
```

Les paramètres peuvent être spécifiés dans le fichier **/etc/modules.conf** afin d'être utilisés automatiquement lors du chargement du module. Dans le cas du noyau 2-6, on utilise le fichier **/etc/conf.modules** :

```
alias eth0 3c59x
options i2c-algo-bit bit_test=1
options bttv card=39
```

La commande **alias** permet de faire une association automatique entre un nom de module générique et le nom de module effectif. Dans le cas présent, le module **3c59x** sera chargé lors de l'initialisation de l'interface réseau Ethernet qui nécessite le module générique **eth0**. Après chaque modification du fichier, on doit utiliser la commande **/sbin/depmod -a**.

Pour décharger le module, on utilise la commande **rmmod** :

```
# rmmod hello
# dmesg | tail -1
Goodbye cruel world!
```

Dans le cas de modules dépendant d'autres modules, on ne peut cependant pas utiliser **insmod**. Reprenons l'exemple du module **vfat** qui nécessite la présence du module **fat** :

```
# insmod vfat
Using /lib/modules/2.4.7-10/kernel/fs/vfat/vfat.o
/lib/modules/2.4.7-10/kernel/fs/vfat/vfat.o: unresolved symbol
➥ fat_read_super_R12852204
/lib/modules/2.4.7-10/kernel/fs/vfat/vfat.o: unresolved symbol
➥ fat_mark_buffer_dirty_R2f222951
...
```

Dans ce cas-là, on devra utiliser la commande **modprobe** qui chargera le module, ainsi que tous les modules dépendants :

```
# modprobe vfat
# lsmod | grep fat
vfat                    9540   0  (unused)
fat                    30688   0  [vfat]
```

De ce fait, il n'est pas possible de décharger le module **fat** car celui-ci est maintenant utilisé par le module **vfat** :

```
# rmmod fat
fat: Périphérique ou ressource occupé
```

On devra tout d'abord décharger **vfat**, puis **fat** :

```
# rmmod vfat
# rmmod fat
```

On se rend vite compte que ces manipulations sont fastidieuses, parfois inextricables si le nombre de dépendances entre les modules devient important. C'est la raison pour laquelle le noyau Linux permet d'utiliser des systèmes automatiques de chargement et déchargement des modules en fonction des besoins. Il existe deux systèmes équivalents pour effectuer cette tâche.

Le démon **kerneld** est un programme standard qui tourne dans l'espace utilisateur et non dans l'espace du noyau. Il utilise un système d'IPC System V afin de communiquer avec le noyau et déterminer en temps réel les modules à charger lorsque l'utilisateur désire activer un nouveau périphérique comme un système de fichier, un périphérique SCSI ou une carte son. Les IPC furent introduits dans Unix System V comme un ensemble de facilités de communications entre divers processus. On peut donc utiliser des fonctionnalités de sémaphores, des zones de mémoire partagée (*shared memory*) ou des files de messages.

Le démon **kerneld** se charge d'effectuer le chargement des modules nécessaires en effectuant les actions équivalentes à la commande **modprobe**. De même, lorsque le module devient inutile parce que le périphérique n'est plus utilisé, **kerneld** effectue automatiquement une action équivalente à la commande **rmmod**. L'indication de la part d'utilisation d'un module est effectuée par le noyau en positionnant le drapeau d'état AUTOCLEAN sur le module. Ce drapeau est visible lors de l'utilisation de la commande **lsmod** :

```
# lsmod
Module                  Size  Used by
hello                    324   0  (unused)
nfsd                  162752   8  (autoclean)
```

Le délai de déchargement du module par **kerneld** est de façon typique de 60 secondes.

Pour diverses raisons, le démon **kerneld** a cependant été remplacé par **kmod**, qui est un *thread* au niveau noyau Linux et qui fonctionne donc dans l'espace noyau et non dans l'espace utilisateur. Les raisons du changement évoquées par les développeurs du noyau sont les suivantes :

- le démon **kerneld** se servait des IPC System V dont l'utilisation n'est pas recommandée, surtout en cas de dialogue avec le noyau ;
- **kmod** et **kerneld** apportent quasiment les même fonctionnalités, en l'occurrence l'appel de **modprobe** ;
- la suppression du code inhérent à **kerneld** dans le noyau a permis de gagner un nombre de lignes de code non négligeable ;
- vu que **kmod** est exécuté dans l'espace du noyau et non dans l'espace utilisateur, il utilise le mécanisme standard des traces d'erreurs du noyau Linux.

kmod et **kerneld** se différencient en ceci que le déchargement automatique du module est absent dans le cas de **kmod**. Le déchargement des modules marqués du drapeau *autoclean* devra être programmé par une ligne **crontab** du super-utilisateur *root* :

```
0-59/5 * * * * /sbin/rmmod -a
```

Le système de fichier /proc

Pour communiquer avec l'espace utilisateur, le noyau Linux utilise un concept emprunté à Unix System V : le système de fichier **/proc**. À la différence des systèmes de fichiers classiques qui sont associés à des périphériques réels, le système de fichier **/proc** est virtuel. Sa structure de système de fichier en fait une représentation facile pour manipuler des paramètres du noyau Linux. On peut en effet utiliser les commandes standards de manipulation des fichiers classiques, ainsi que la redirection des entrées/sorties très utilisée sous Linux.

En particulier, la commande **lsmod** de lecture des modules chargés n'est en fait qu'un raccourci pour la visualisation du fichier virtuel **/proc/modules** :

```
# cat /proc/modules
hello                     324   0 (unused)
nfsd                   162752   8 (autoclean)
lockd                   44080   1 (autoclean) [nfsd]
```

De même, on peut visualiser les paramètres standards du système comme la mémoire disponible au moyen de **/proc/meminfo**, la version du noyau avec **/proc/version**, le type de processeur utilisé avec **/proc/cpuinfo**, ou les systèmes de fichiers supportés par le noyau avec **/proc/filesystems**. Cette liste n'est, bien entendu, pas exhaustive car un pilote de périphérique peut ajouter dynamiquement des fichiers et répertoires à **/proc** lors du chargement du module associé.

De même, les valeurs numériques présentes dans **/proc** représentent les zones d'information des processus courant, chaque valeur correspondant au PID (*Processus IDentifier*) du processus en question. Ces sous-répertoires contiennent les informations propres au processus en question :

```
# ls -l /proc/25832
total 0
-r--r--r--    1 root     root            0 oct 21 23:42 cmdline
-r--r--r--    1 root     root            0 oct 21 23:42 cpu
lrwx------    1 root     root            0 oct 21 23:42 cwd -> /
-r--------    1 root     root            0 oct 21 23:42 environ
lrwx------    1 root     root            0 oct 21 23:42 exe -> /usr/sbin/httpd
dr-x------    2 root     root            0 oct 21 23:42 fd
pr--r--r--    1 root     root            0 oct 21 23:42 maps
-rw-------    1 root     root            0 oct 21 23:42 mem
lrwx------    1 root     root            0 oct 21 23:42 root -> /
-r--r--r--    1 root     root            0 oct 21 23:42 stat
-r--r--r--    1 root     root            0 oct 21 23:42 statm
-r--r--r--    1 root     root            0 oct 21 23:42 status
```

Par exemple, le fichier **status** contient des informations sur l'état du processus en question :

```
# cat /proc/25832/status
Name:   httpd
State:  S (sleeping)
Pid:    25832
PPid:   2173
...
```

Le système de fichier **/proc** est également utilisable en écriture, ce qui permet de modifier dynamiquement le comportement du noyau Linux sans aucune compilation. Un exemple classique est la validation d'option comme par le transfert de paquets IP (*IP forwarding*). On peut connaître la valeur de l'option d'*IP forwarding* en faisant :

```
# cat /proc/sys/net/ipv4/ip_forward
1
```

Le système retourne la valeur 1, ce qui signifie que l'*IP forwarding* est validé. On peut l'inhiber en faisant simplement :

```
# echo 0 > /proc/sys/net/ipv4/ip_forward
```

Qui a dit que Linux était compliqué ?

Une description complète du pseudo-système de fichiers **/proc** est disponible dans le répertoire de documentation des sources du noyau Linux, soit :

```
/usr/src/linux/Documentation/proc.txt
```

pour le noyau 2.2, et

```
/usr/src/linux-2.4/Documentation/filesystems/proc.txt
```

pour les noyaux 2.4 et 2.6.

Compilation du noyau

Les distributions classiques fournissent des noyaux binaires pré-compilés, soit sous forme d'archives RPM dans le cas des distributions Red Hat, Mandriva et SuSE, soit sous forme d'archives DEB dans le cas de la distribution. DEBIAN.

L'utilisation de Linux dans un environnement industriel embarqué obligera cependant à adapter le noyau à l'environnement matériel, soit en partie l'objet même de cet ouvrage !

Notre intention dans cette section est de fournir les éléments nécessaires à l'obtention de l'archive officielle du noyau, à l'extraction de celle-ci, la configuration de ce noyau, puis sa compilation et son installation. Ces éléments seront donnés sous un angle général sans considération de réduction de la taille du noyau. Les possibilités de réduction seront explicitées dans les chapitres suivants.

Comme dans tous les développements Open Source, la distribution officielle du noyau Linux est fournie sous forme d'archive au format tar compressée au format gzip ou bzip2. Ces archives sont disponibles auprès du serveur *ftp://ftp.kernel.org* ou l'un de ses miroirs tel le site *ftp://ftp.free.fr*.

Les sources du noyau Linux sont également fournies par les distributions classiques sous forme d'archives RPM ou DEB. Nous devons insister ici sur le fait que l'archive du noyau fournie par exemple sur le CD-Rom Red Hat n'est pas identique à celle que vous trouverez sur le site *ftp://ftp.kernel.org*.

La licence GPL permet en effet à l'éditeur d'apporter des modifications aux sources du noyau à condition que ces modifications soient également fournies en sources sur le CD-Rom.

De plus, il peut arriver que certains pilotes de périphériques ne soient par fournis sur l'archive officielle du noyau Linux, étant considérés par Linus Torvalds comme pas assez aboutis, bien qu'ils soient présents sur le CD-Rom d'un éditeur souvent en raison de la pression commerciale.

Attention

Dans la suite de l'ouvrage, et ce afin de ne pas entrer dans le jeu de la guerre des distributions, nous ferons toujours référence au noyau Linux officiel et non à ceux fournis par les éditeurs.

Nous donnons ci-après la trace du dialogue ftp avec le site *ftp://ftp.free.fr* afin de récupérer les sources du noyau officiel. Le client ftp utilisé est l'excellent **ncftp** disponible dans toutes les bonnes distributions.

```
$ ncftp ftp.free.fr
NcFTP 3.1.5 (Oct 13, 2002) by Mike Gleason (ncftp@ncftp.com).
Connecting to 213.228.0.141...
Welcome to ProXad FTP server
Logging in...
Login successful.
Logged in to ftp.free.fr.
ncftp / > cd mirrors/ftp.kernel.org/linux/kernel
Directory successfully changed.
ncftp ...ernel.org/linux/kernel > dir
-r--r--r--    1 1000     1000        18458 mar 13  1994 COPYING
-r--r--r--    1 1000     1000        36981 sep 16  1996 CREDITS
drwxr-sr-x    5 1000     1000         4096 nov 24  2001 crypto
drwxr-sr-x    4 1000     1000         4096 mar 20  2003 Historic
drwxr-sr-x   61 1000     1000         4096 avr  8 23:07 people
drwxr-sr-x    6 1000     1000         4096 mar 13  2003 ports
drwxr-sr-x    3 1000     1000         4096 sep 16  2000 projects
-r--r--r--    1 1000     1000        12056 sep 16  1996 README
drwxr-sr-x    2 1000     1000         4096 avr 14  2000 SillySounds
drwxr-sr-x    3 1000     1000         4096 fév 14  2002 testing
drwxr-sr-x    2 1000     1000         4096 mar 20  2003 uemacs
drwxr-sr-x    2 1000     1000         4096 mar 20  2003 v1.0
drwxr-sr-x    2 1000     1000        20480 mar 20  2003 v1.1
drwxr-sr-x    2 1000     1000         8192 mar 20  2003 v1.2
drwxr-sr-x    2 1000     1000        36864 mar 20  2003 v1.3
drwxr-sr-x    3 1000     1000        12288 fév  8  2004 v2.0
drwxr-sr-x    2 1000     1000        45056 mar 20  2003 v2.1
drwxr-sr-x    3 1000     1000         8192 mar 24  2004 v2.2
drwxr-sr-x    2 1000     1000        20480 mar 20  2003 v2.3
drwxr-sr-x    5 1000     1000        12288 avr  4 01:50 v2.4
drwxr-sr-x    4 1000     1000        28672 jui 14  2003 v2.5
drwxr-sr-x    6 1000     1000         8192 avr  7 19:30 v2.6
```

Ce miroir conserve la trace de toutes les versions du noyau Linux depuis la 1.0. Nous rappelons que la numérotation des versions du noyau suit les règles suivantes :

- Le noyau est identifié par un triplet *version.révision.patch*, exemple 2.4.13.

- Les révisions paires identifient des noyaux stables. En toute rigueur, ce sont les seuls qu'il faut utiliser dans le cas de produits industriels.
- Les corrections (ou *patch*) sur un noyau stable n'incluent que des corrections de bogues mais pas de nouvelles fonctionnalités.
- Les révisions impaires sont des noyaux de développement. La fréquence de diffusion de ces noyaux peut être très élevée, parfois un par semaine ou plus. Certaines corrections peuvent apporter de nouvelles fonctionnalités. Certains noyaux de développement sont très stables mais leur utilisation n'est absolument pas recommandée pour des applications industrielles.
- Le passage d'une version de développement à une version stable suppose un gel du code ou *code-freeze* décidé par Linus lui-même. Ce passage peut connaître des phases intermédiaires comme les pré-versions stables, exemple 2.3.99-pre1 à pre9. Cela tient à ce que la complexité du noyau va croissant au cours du temps et que la diffusion d'une première version stable est une opération de plus en plus délicate.

Si nous souhaitons récupérer la dernière version du noyau 2.4, nous devons procéder aux actions suivantes :

```
ncftp ...ernel.org/linux/kernel > cd v2.4
ncftp ....org/linux/kernel/v2.4 > ls LATEST*
LATEST-IS-2.4.13
```

La deuxième ligne permet de connaître la dernière version du noyau, ici en l'occurrence 2.4.13. Il suffit ensuite de récupérer l'archive soit au format *gzip*, soit au format *bzip2*. Ce dernier format est recommandé car il permet de réduire la taille de l'archive, et donc le temps de téléchargement. Ici, l'on voit que l'on gagne 5 Mo sur la taille de l'archive :

```
ncftp ....org/linux/kernel/v2.4 > dir linux-2.4.13*
-rw-r--r--   1 daemon   daemon     23111925 oct 24 05:28 linux-2.4.13.tar.bz2
-rw-r--r--   1 daemon   daemon          248 oct 24 05:28 linux-2.4.13.tar.bz2.sign
-rw-r--r--   1 daemon   daemon     28561727 oct 24 05:28 linux-2.4.13.tar.gz
-rw-r--r--   1 daemon   daemon          248 oct 24 05:28 linux-2.4.13.tar.gz.sign
ncftp ....org/linux/kernel/v2.4 > get linux-2.4.13.tar.bz2
linux-2.4.13.tar.bz2:                                  22,04 MB   44,23 kB/s
ncftp ....org/linux/kernel/v2.4 >
```

Après récupération de l'archive, il faut la décompresser dans le répertoire adéquat. Ce répertoire est généralement **`/usr/src`**. Il se peut que vous disposiez déjà de l'arborescence des sources grâce à votre distribution Linux, soit du fait de l'installation préalable d'un noyau 2.4 :

```
# cd /usr/src
# ls -ld linux*
lrwxrwxrwx    1 root     root           11 oct 27 22:02 linux-2.4 -> linux-2.4.2
drwxr-xr-x   16 root     root         4096 oct 22 14:42 linux-2.4.2
```

ou bien :

```
# ls -ld linux*
lrwxrwxrwx    1 root     root           11 oct 27 22:02 linux -> linux-2.4.2
drwxr-xr-x   16 root     root         4096 oct 22 14:42 linux-2.2.19
```

Si l'on regarde l'archive téléchargée, on remarque que celle-ci est construite à partir du répertoire **/usr/src** :

```
# tar tjvf linux-2.4.13.tar.bz2
drwxr-xr-x torvalds/eng       0 2001-10-24 07:21:20 linux/
-rw-r--r-- torvalds/eng   16909 2001-10-24 07:21:20 linux/Makefile
drwxr-xr-x torvalds/eng      0 2001-10-24 07:17:55 linux/fs/
-rw-r--r-- torvalds/eng    9564 2001-09-14 01:04:43 linux/fs/stat.c
-rw-r--r-- torvalds/eng    2292 2001-10-05 00:13:18 linux/fs/Makefile
-rw-r--r-- torvalds/eng   8859 2001-08-05 22:12:41 linux/fs/read_write.c
...
```

On notera l'option « j » de la commande **tar** qui permet d'utiliser le décompresseur **bzip2** avant d'extraire l'archive.

S'il existe déjà un répertoire **/usr/src/linux**, il faudra donc le renommer si l'on ne veut pas l'écraser avec les fichiers de la nouvelle archive. Cette étape n'est plus nécessaire dans le cas des dernières versions du noyau 2.4 (à partir de la 2.4.20) et du noyau 2.6 car le répertoire créé par l'extraction du noyau est – enfin – indexé sur la version du noyau.

Dans notre cas, il suffit d'extraire l'archive dans **/usr/src** :

```
# tar xjvf /root/linux-2.4.13.tar.bz2  | more
linux/
linux/Makefile
linux/fs/
linux/fs/stat.c
linux/fs/Makefile
linux/fs/read_write.c
linux/fs/block_dev.c
...
#
```

ce qui donne sur le répertoire :

```
# ls -ld linux*
drwxr-xr-x   14 1046     101          4096 oct 27 22:18 linux
lrwxrwxrwx    1 root     root           11 oct 27 22:02 linux-2.4 -> linux-2.4.2
drwxr-xr-x   16 root     root         4096 oct 22 14:42 linux-2.4.2
```

puis de renommer le répertoire **linux** avec la version correcte, soit 2.4.13, et finalement de positionner le lien symbolique sur cette version, ce qui donne :

```
# mv linux linux-2.4.13
[root@localhost src]# rm -f linux-2.4
[root@localhost src]# ln -s linux-2.4.13 linux-2.4
[root@localhost src]# ls -ld linux-2.4*
lrwxrwxrwx    1 root     root           12 oct 27 22:20 linux-2.4 -> linux-2.4.13
drwxr-xr-x   14 1046     101          4096 oct 27 22:18 linux-2.4.13
drwxr-xr-x   16 root     root         4096 oct 27 22:20 linux-2.4.2
```

Cette méthode permet de conserver les différentes arborescences des sources du noyau et de passer de l'une à l'autre en positionnant un simple lien symbolique.

On peut alors se positionner dans le répertoire afin de regarder la structure globale de l'arborescence :

```
# cd linux-2.4
# ls -l
total 240
drwxr-xr-x   17 1046     101          4096 fév 13  2001 arch
-rw-r--r--    1 1046     101         18689 oct 10 00:00 COPYING
-rw-r--r--    1 1046     101         77588 oct 21 19:20 CREDITS
drwxr-xr-x   28 1046     101          4096 oct 27 22:18 Documentation
drwxr-xr-x   38 1046     101          4096 oct 27 22:18 drivers
drwxr-xr-x   41 1046     101          4096 oct 27 22:17 fs
drwxr-xr-x   24 1046     101          4096 oct 24 07:18 include
drwxr-xr-x    2 1046     101          4096 oct 27 22:17 init
drwxr-xr-x    2 1046     101          4096 oct 27 22:17 ipc
drwxr-xr-x    2 1046     101          4096 oct 27 22:17 kernel
drwxr-xr-x    2 1046     101          4096 oct 27 22:17 lib
-rw-r--r--    1 1046     101         38094 oct 22 17:37 MAINTAINERS
-rw-r--r--    1 1046     101         16909 oct 24 07:21 Makefile
drwxr-xr-x    2 1046     101          4096 oct 27 22:17 mm
drwxr-xr-x   27 1046     101          4096 oct 27 22:17 net
-rw-r--r--    1 1046     101         14242 oct  5 21:10 README
-rw-r--r--    1 1046     101          2815 avr  6  2001 REPORTING-BUGS
-rw-r--r--    1 1046     101          8884 mar  7  2001 Rules.make
drwxr-xr-x    5 1046     101          4096 oct 27 22:18 scripts
```

Dans le cas du noyau 2.6, la liste des répertoires est très similaire :

```
# cd /usr/src/linux-2.6.10
# ls -l
total 408
drwxrwxr-x   24 root     root         4096 déc 24 22:33 arch
-rw-r--r--    1 root     root        18691 déc 24 22:34 COPYING
-rw-r--r--    1 root     root        88167 déc 24 22:35 CREDITS
drwxrwxr-x    2 root     root         4096 avr  6 07:55 crypto
drwxrwxr-x   46 root     root         4096 déc 24 22:35 Documentation
drwxrwxr-x   46 root     root         4096 avr  6 07:55 drivers
drwxrwxr-x   54 root     root         4096 avr  6 07:55 fs
drwxrwxr-x   37 root     root         4096 fév 11 09:26 include
drwxrwxr-x    2 root     root         4096 avr  6 07:55 init
drwxrwxr-x    2 root     root         4096 avr  6 07:55 ipc
drwxrwxr-x    4 root     root         4096 avr  6 07:55 kernel
drwxrwxr-x    5 root     root         4096 avr  6 07:55 lib
-rw-r--r--    1 root     root        55119 déc 24 22:35 MAINTAINERS
-rw-r--r--    1 root     root        43257 fév 11 09:26 Makefile
drwxrwxr-x    2 root     root         4096 avr  6 07:55 mm
-rw-r--r--    1 root     root       104306 fév 11 09:44 Module.symvers
drwxrwxr-x   32 root     root         4096 avr  6 07:55 net
-rw-r--r--    1 root     root        13970 déc 24 22:35 README
-rw-r--r--    1 root     root         2815 déc 24 22:34 REPORTING-BUGS
drwxrwxr-x    9 root     root         4096 avr  6 07:55 scripts
drwxrwxr-x    4 root     root         4096 avr  6 07:55 security
drwxrwxr-x   15 root     root         4096 avr  6 07:55 sound
drwxrwxr-x    2 root     root         4096 avr  6 07:55 usr
```

Nous pouvons donner une brève description des fichiers et sous-répertoires de l'arborescence du noyau. La description de certains sous-répertoires sera approfondie si nécessaire dans la suite de l'ouvrage.

- `arch` contient le code source spécifique des architectures matérielles comme x86, ppc, alpha ou m68k.
- `Documentation` contient des fichiers de documentation au format.
- `drivers` contient l'arborescence des divers pilotes de périphériques.
- `fs` contient le code source des différents systèmes de fichiers, ou *file-systems* supportés par le noyau. Nous pouvons citer par exemple *ext2*, *vfat* ou *iso9660*.
- `include` contient les fichiers d'en-tête C nécessaires à la compilation du noyau mais également au développement d'applications.
- `init` contient le fichier principal `main.c`, contenant la fonction principale `main()`, du noyau Linux.
- `ipc` contient le code source de la version des IPC System V du noyau Linux.
- `kernel` contient le code source des fonctions majeures du noyau comme la gestion des tâches ou des processus.
- `mm` contient les fonctions de gestion de la mémoire ou *memory-management*.
- `net` contient le code source des différents protocoles réseau supportés par le noyau Linux. Nous pouvons citer i*pv4*, *ipv6* ou *x25*.
- `scripts` contient le code source des outils de configuration du noyau.
- `Makefile` et `Rules.make` sont les fichiers utilisés par la commande *make* lors de la compilation du noyau.
- `sound` contient dans le cas du noyau 2.6 les pilotes ALSA (Advanced Linux Sound Archicture) désormais intégrés en standard aux sources du noyau.

La génération d'un nouveau noyau passe par les phases suivantes :

- la configuration des options du noyau ;
- la génération des dépendances (uniquement en 2.4) ;
- la compilation du noyau statique ;
- la compilation des modules, si nécessaire.

Toutes les phases de génération du noyau passent par l'utilisation de la commande `make`, largement utilisée dans tous les développements Unix et même sur d'autres systèmes d'exploitation. Le principe de cet utilitaire est de comparer la date des fichiers résultats par rapport à celle des fichiers sources en fonction de règles définies dans un fichier de commande nommé par défaut `Makefile`. Cela permet de générer uniquement les fichiers résultats dont les sources ont été modifiées depuis la dernière génération.

La configuration a pour objet de permettre de choisir parmi les multiples possibilités du noyau les fonctionnalités que l'on désire utiliser, par exemple l'utilisation de périphé-

riques SCSI ou le support de cartes ISDN. Dans le cas d'un système embarqué, une configuration optimale, réduite aux strictes fonctionnalités nécessaires, sera indispensable tant au niveau des performances que de la taille du noyau.

La configuration des options du noyau se fait par la commande :

```
# make config
rm -f include/asm
( cd include ; ln -sf asm-i386 asm)
/bin/sh scripts/Configure arch/i386/config.in
#
# Using defaults found in arch/i386/defconfig
#
*
* Code maturity level options
*
Prompt for development and/or incomplete code/drivers (CONFIG_EXPERIMENTAL) [N/y/?]
```

Ce mode de configuration permet seulement une définition séquentielle en mode texte des diverses options du noyau, ce qui est très difficilement utilisable dans les noyaux récents. En particulier, il n'est pas possible de revenir en arrière pour modifier une option. On préférera aujourd'hui la configuration pleine page du **`make menuconfig`** utilisant la bibliothèque *Curses* ou bien la version *X Window* **`make xconfig`** recourant au langage de script *Tcl/Tk*. Dans le cas du noyau 2.6, la configuration graphique **`make xconfig`** est désormais accessible à travers un programme basé sur Qt.

La figure 4-2 ci-après indique l'écran de dialogue obtenu lors du lancement de la commande **`make menuconfig`** :

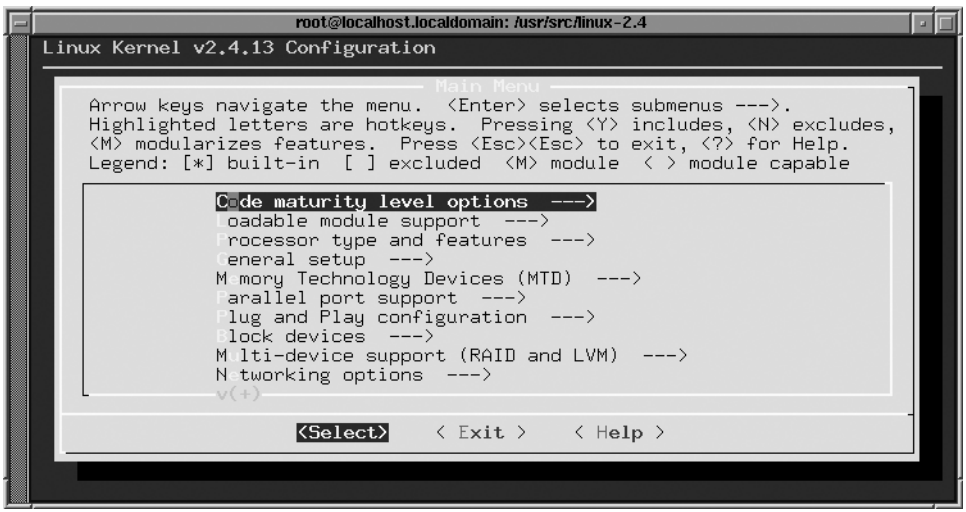


Figure 4-2

Make menuconfig.

La figure ci-après indique l'écran de dialogue obtenu lors du lancement de la commande `make xconfig` :

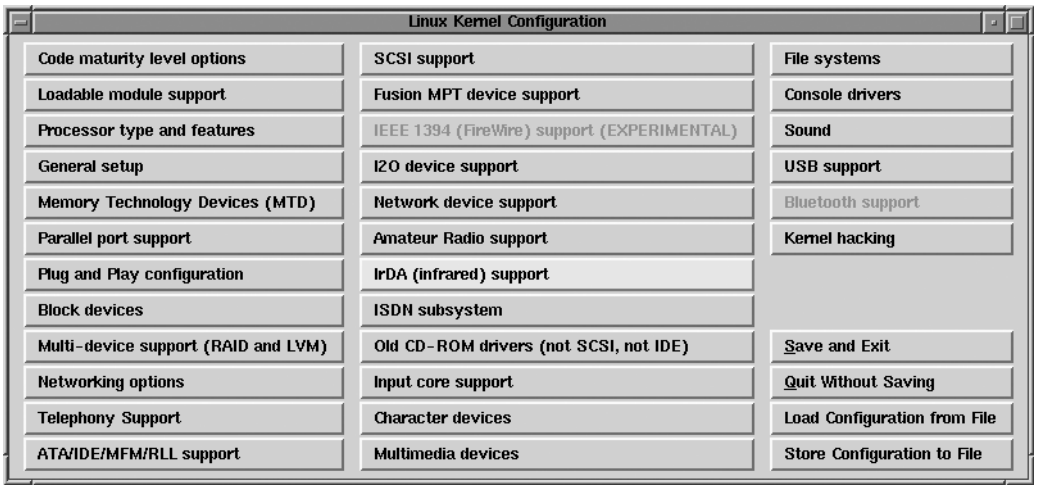


Figure 4-3
Make xconfig pour le noyau 2.4.

Dans le cas du noyau 2.6, on obtient l'écran suivant :

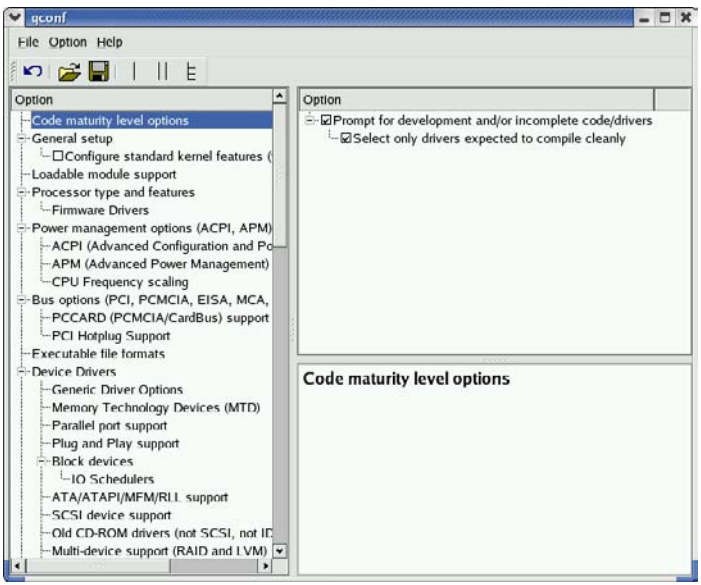


Figure 4-4
Make xconfig pour le noyau 2.6.

Le lecteur averti conviendra que cette dernière solution est de loin la plus efficace ! Ce même lecteur notera que l'aspect de l'écran de configuration du noyau est beaucoup plus inspiré par la langue de Shakespeare que par celle de Pagnol, ce qui signifie qu'il faudra un minimum de maîtrise de l'anglais technique pour s'attaquer à cette configuration.

L'écran obtenu en cliquant sur le premier bouton *Code maturity level options* est très important car il conditionne l'affichage des écrans de configuration des fonctionnalités les plus récentes. Si vous désirez utiliser ces fonctionnalités, vous devrez obligatoirement valider cette option comme indiqué ci-après sur la figure suivante :

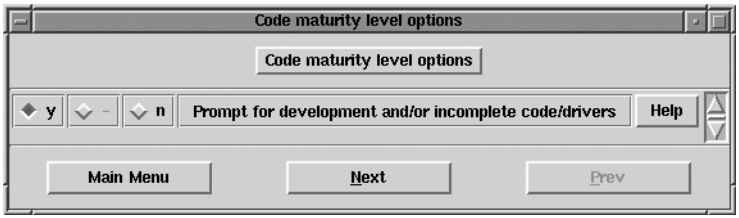


Figure 4-5

Code maturity level options.

En actionnant le bouton *Help*, on obtient un écran d'aide affiché à partir du répertoire de documentation cité précédemment. Le symbole affiché en titre de la page d'aide, soit `CONFIG_EXPERIMENTAL` dans ce cas, sera utilisé comme variable logique dans le fichier résultat de la configuration décrit plus bas dans cette section.

Chaque variable logique pourra prendre deux ou trois états selon le type de configuration :

- Les configurations de type oui/non prendront les valeurs *y/n*. C'est le cas dans l'exemple précédent.
- Les configurations concernant le support d'un protocole ou d'un périphérique prendront les valeurs *y/n/m*, le dernier état correspondant au support par un module chargeable au lieu d'un support compilé dans la partie statique du noyau.

Un exemple de ce type de configuration est donné sur la figure 4-6 ci-après :

On notera dans cet exemple que la validation ou non d'une option peut entraîner l'accès ou non à d'autres options de configuration.

Lorsque la configuration est terminée, celle-ci doit être sauvegardée grâce au bouton *Save and exit* situé en bas à droite de l'écran principal. Par défaut, la configuration courante sera sauvegardée sur le fichier `.config` du répertoire des sources du noyau :

```
# ls -la .config
-rw-r--r--    1 root     root        15832 oct 28 00:17 .config
```

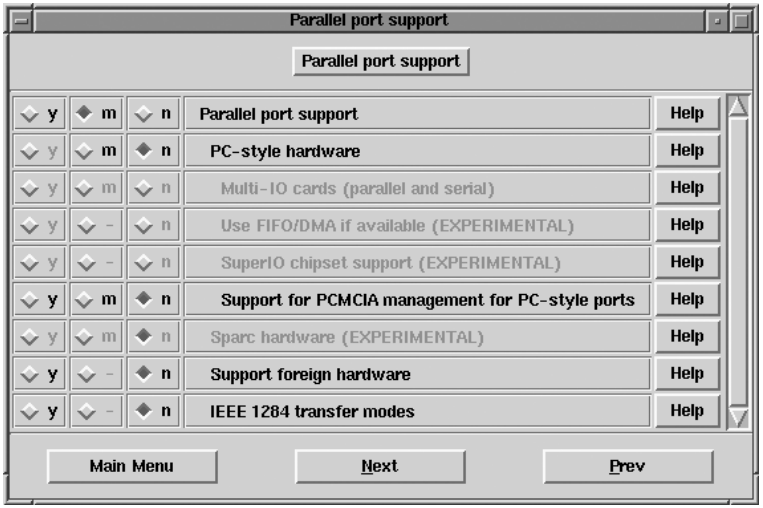


Figure 4-6

Parallel port support, exemple de configuration à 3 états.

En éditant le fichier, on peut vérifier la validation des options, comme le support des développements récents ou du port parallèle en module :

```
#
# Code maturity level options
#
CONFIG_EXPERIMENTAL=y
...
#
# Parallel port support
#
CONFIG_PARPORT=m
# CONFIG_PARPORT_PC is not set
# CONFIG_PARPORT_PC_PCMCIA is not set
# CONFIG_PARPORT_AMIGA is not set
# CONFIG_PARPORT_MFC3 is not set
# CONFIG_PARPORT_ATARI is not set
# CONFIG_PARPORT_SUNBPP is not set
# CONFIG_PARPORT_OTHER is not set
# CONFIG_PARPORT_1284 is not set
```

L'utilitaire de configuration permet également de sauvegarder la configuration sous un nom différent – *Store configuration to file* – ainsi que de charger une configuration à partir d'un fichier au format présenté plus haut – *Load configuration from file*.

Notez bien

Ces possibilités seront particulièrement utiles dans le cas de tests successifs lors de l'optimisation d'un noyau Linux embarqué.

La phase suivante est suggérée par l'utilitaire de configuration lors de la sauvegarde comme indiqué sur la figure suivante :

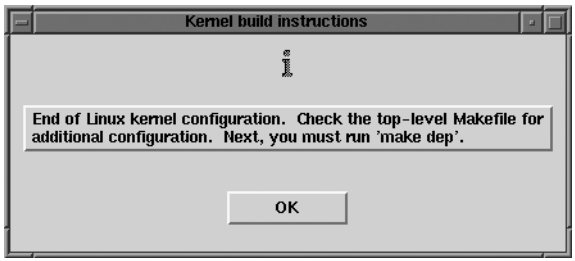


Figure 4-7

Sauvegarde de la configuration.

La boîte de dialogue indique à juste titre qu'il est possible de modifier le fichier **Makefile** avant de lancer les phases suivantes. En particulier, il peut être très intéressant de modifier le paramètre **EXTRAVERSION** situé en tête du fichier **Makefile** :

```
# head Makefile
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 13
EXTRAVERSION = -pf_test1
```

Comme indiqué au début du chapitre, la modification de ce paramètre va permettre de faire cohabiter plusieurs versions identiques du noyau, correspondant à des tests successifs. En particulier, les arborescences des modules seront suffixées par la valeur de l'**EXTRAVERSION**, ce qui donnera dans notre cas **/lib/modules/2.4.13-pf_test1**.

Lorsque le fichier de configuration est au point, on doit créer les dépendances des fichiers **Makefile** des différents sous-répertoires. Le fichier de configuration du noyau étant unique, le but de cette action est de propager les options choisies dans les sous-répertoires des sources du noyau. Pour cela, on doit effectuer un

```
make dep
```

comme indiqué par la boîte de dialogue de la figure 4-7. Cette étape n'est pas nécessaire dans le cas du noyau 2.6. Si on l'effectue en environnement 2.6, on obtient le message suivant :

```
# make dep
*** Warning: make dep is unnecessary now.
```

La compilation de la partie statique du noyau par un simple :

```
make
```

générera le noyau sous forme non compressée sous le nom **vmlinux** dans le répertoire **arch/i386/boot**. Pour des raisons évidentes de taille du fichier noyau, on préférera la commande :

```
make bzImage
```

qui générera un fichier compressé **bzImage** au même endroit. Le noyau contient lui-même le code de décompression utilisé lors du démarrage du système.

À la fin de la compilation, on obtient le fichier noyau ainsi que le fichier **System.map** qui renferme la liste des adresses internes du noyau :

```
# ls -l System.map
-rw-r--r--    1 root     root       438251 oct 28 01:03 System.map
# ls -l arch/i386/boot/bzImage
-rw-r--r--    1 root     root       848556 oct 28 01:03 arch/i386/boot/bzImage
```

On doit ensuite copier ces fichiers dans le répertoire **/boot** contenant les noyaux du système, et ce en respectant la valeur de l'**EXTRAVERSION** :

```
# cp arch/i386/boot/bzImage /boot/bzImage-2.4.13_pf_test1
# cp System.map /boot/System.map-2.4.13-pf_test1
```

La phase suivante est la génération des modules chargeables qui représentent la partie dynamique du noyau. Pour cela, on fait :

```
make modules
```

puis pour installer les modules :

```
make modules_install
```

Astuce !

Lorsque la compilation du noyau n'aura plus de secrets pour vous, vous pourrez de façon avantageuse enchaîner ces commandes en séparant les différentes étapes par un point-virgule : make dep; make clean; etc.

De même, le résultat de la compilation peut être redirigé sur un fichier de trace en utilisant la syntaxe `1>fichier_trace 2>&1 &` à la fin de l'enchaînement de commandes `make`.

La génération du nouveau noyau est alors terminée ; il reste à ajouter ce noyau dans le système de démarrage comme nous allons le voir dans la section suivante.

Configuration du démarrage

Pour le démarrage ou *boot* d'un système, on recourt le plus souvent à un programme de chargement ou *loader*. Il existe divers programmes de chargement mais le plus utilisé est certainement LILO, le *LInux LOader*. Le principe de LILO est similaire à celui des autres programmes de démarrage, à savoir installer un programme de chargement du système d'exploitation dans les 512 premiers octets du périphérique de démarrage, le plus souvent d'un disque dur.

LILO est configurable par un fichier unique **lilo.conf** situé sur le répertoire **/etc**.

La plupart des distributions génèrent un fichier de configuration similaire à celui décrit ci-après :

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
default=linux

image=/boot/vmlinuz-2.4.2-2
        label=linux
        read-only
        root=/dev/hda3
```

Le paramètre **boot** indique à LILO qu'il devra installer son secteur de démarrage (ou *boot sector*) sur le premier secteur du premier disque dur. C'est le cas le plus fréquemment utilisé, ce qui implique que LILO contrôle le démarrage de tous les systèmes d'exploitation de la machine en cas d'installation multi-système.

Le paramètre **map** est automatique généré par LILO.

Le paramètre **install** indique à LILO le nom du nouveau fichier de secteur de *boot* à installer. Ce fichier contient le code de démarrage à installer sur le premier secteur du disque. Le mot-clé **read-only** indique qu'il faut initialement monter le système de fichier en lecture seule, et ce en cas de vérification d'intégrité du système par la commande ***e2fsck***. Cette vérification a lieu si le système n'a pas été arrêté correctement, mais également à intervalles réguliers en fonction du nombre de montages du système de fichier au cours du temps.

Le paramètre **image** décrit un système d'exploitation à démarrer, en l'occurrence un noyau Linux à charger. Les sous-paramètres **label** et **root** indiquent respectivement le nom symbolique à utiliser pour désigner ce noyau, ainsi que la partition principale utilisée par ce dernier.

Les paramètres **prompt** et **timeout** indiquent que LILO interrompra son chargement durant quelques dixièmes de secondes (ici 50) de manière à laisser le temps à l'utilisateur d'entrer le nom d'une image comme cela a été précédemment défini grâce au sous-paramètre **label**. Si l'utilisateur ne fait aucune action, l'image décrite par le paramètre **default** sera chargée. Pendant ce temps de latence, l'utilisateur peut également entrer des paramètres qui modifieront le comportement du démarrage du système. Pour forcer le système à démarrer au niveau 1 (mono-utilisateur ou *single user*), on entrera **linux 1** à l'invité LILO. Pour forcer la valeur de la taille de mémoire vive à 128 Mo, on entrera **MEM=128M**. En cas d'utilisation systématique, ces paramètres peuvent être spécifiés dans le fichier **/etc/lilo.conf** grâce à la directive **append** :

```
append="MEM=128M"
```

L'adjonction d'un nouveau noyau est très simple car il suffit d'ajouter des lignes du type :

```
image=/boot/bzImage-2.4.13-pf_test1
        label=new
        read-only
        root=/dev/hda3
```

ce qui permettra de charger l'image nommée *new* lors du démarrage du système.

Dans le cas où le démarrage du système nécessite le chargement d'un pilote non intégré dans la partie statique du noyau, on devra utiliser un disque mémoire ou *ramdisk* initial. Ce disque mémoire sera chargé par un fichier généré à partir du noyau par la commande **mkinitrd** :

```
# mkinitrd initrd-2.4.13-pf_test1.img 2.4.13-pf_test1
# ls -l initrd-2.4.13
-rw-r--r--    1 root     root       251444 oct 28 02:04 initrd-2.4.13-pf_test1.img
```

La validation du nouveau paramétrage et l'écriture du nouveau secteur de démarrage se font par la commande :

```
# /sbin/lilo  -v
LILO version 21.4-4, Copyright (C) 1992-1998 Werner Almesberger
'lba32' extensions Copyright (C) 1999,2000 John Coffman

Reading boot sector from /dev/hda
Merging with /boot/boot.b
Mapping message file /boot/message
Boot image: /boot/vmlinuz-2.4.2-2
Added linux *
Boot image: /boot/bzImage-2.4.13-pf_test1
Added new
```

Très important !

L'utilisation de la commande susmentionnée est indispensable après chaque modification du noyau. En cas d'omission, le système pourrait ne plus être en mesure de redémarrer. Dans tous les cas, nous vous conseillons de construire une disquette de démarrage comme cela est indiqué ci-après.

Si l'on ne désire pas modifier le secteur de démarrage du disque, le programme LILO peut également être installé sur une disquette construite grâce à la commande **mkbootdisk** sur distribution Red Hat ou **mkboot** sur distribution DEBIAN :

```
mkbootdisk --device /dev/fd0 2.4.13-pf_test1
```

Il est également possible de configurer le démarrage sans utiliser LILO ni aucun programme de chargement. Pour cela, il suffit de copier la partie statique du noyau Linux sur une disquette formatée par un simple :

```
cp /boot/bzImage-2.4.13_pf_test1 /dev/fd0
```

puis d'indiquer à ce noyau ses différents paramètres de démarrage en utilisant la commande **rdev** :

```
# rdev --help
usage: rdev [ -rsv ] [ -o OFFSET ] [ IMAGE [ VALUE [ OFFSET ] ] ]
  rdev /dev/fd0  (or rdev /linux, etc.) displays the current ROOT device
  rdev /dev/fd0 /dev/hda2         sets ROOT to /dev/hda2
  rdev -R /dev/fd0 1              set the ROOTFLAGS (readonly status)
  rdev -s /dev/fd0 /dev/hda2      set the SWAP device
  rdev -r /dev/fd0 627            set the RAMDISK size
  rdev -v /dev/fd0 1              set the bootup VIDEOMODE
  rdev -o N ...                   use the byte offset N
  rootflags ...                   same as rdev -R
  swapdev ...                     same as rdev -s
  ramsize ...                     same as rdev -r
  mode vidéo ...                     pareil que rdev -v
Note: les modes vidéo sont : -3=Demander, -2=Etendu, -1=NormalVga, 1=touche1,
  ➥ 2=touche2,...
  utilisez -R 1 pour monter la racine en lecture seule, -R 0 pour lecture/écriture.
```

Ce qui donne dans notre cas :

```
rdev /dev/fd0 /dev/hda1
```

pour indiquer au noyau que son *root filesystem* est situé sur **`/dev/hda1`**, soit la première partition du premier disque IDE, puis :

```
rdev -R /dev/fd0 1
```

pour indiquer de monter initialement ce système de fichier en lecture seule. La disquette ainsi obtenue permet alors de démarrer un système. Cette méthode nécessite cependant la présence des pilotes de périphériques indispensables au *boot* dans la partie *statique* du noyau, soit le fichier **`bzImage`**.

Répertoires et fichiers principaux

Le système d'exploitation Linux est remarquablement bien organisé en ce qui concerne la répartition des fichiers système. La raison principale en est l'héritage des architectures Unix qui, en dépit de leurs différences, surent garder une structure homogène et facilement compréhensible pour un administrateur teinté d'une culture Unix générique. En effet, même si la guerre de Unix fit rage pendant de nombreuses années, il sera relativement simple à un administrateur système *Solaris*, donc *System V*, d'administrer un système FreeBSD ou Linux, la réciproque étant vraie.

La seule difficulté réside le plus souvent dans la connaissance des outils d'administration propriétaires fournis par les éditeurs ou constructeurs afin de faciliter le travail d'administration du système. Sachant qu'un véritable administrateur Unix œuvre exclusivement à l'éditeur de texte **`vi`** ou **`emacs`**, il n'y a pas beaucoup de souci à se faire de ce côté-là.

L'organisation des fichiers d'un système Linux est définie par un document intitulé dans sa version originale le *Filesystem Hierarchy Standard* (*FHS*) que l'on peut traduire par *Standard d'organisation du système de fichiers*.

Ce document n'est pas une norme officielle mais vise à unifier l'organisation des systèmes de fichiers Unix afin de faciliter le passage d'une version d'Unix à une autre. Il est disponible sur Internet à l'adresse *http://www.pathname.com/fhs*.

Comme nous l'avons décrit dans la sous-section concernant le noyau Linux, ce dernier nécessite la présence d'un système de fichiers principal ou racine appelé *root filesystem*. La commande **`mount`** permet de connaître la partition physique associée à ce système de fichier symbolisé par le caractère *slash* (/).

```
# mount
§/dev/hda2 on / type ext2 (rw)
```

D'après le FHS, le système de fichier de Linux est organisé de la manière suivante, du moins pour les entrées principales :

```
/ -- racine du système
     +-bin       Principales commandes utilisateur
     +-boot      Noyaux et chargeurs du système
     +-dev       Pseudo fichiers des pilotes (devices)
     +-etc       Fichiers de configuration
     +-lib       Bibliothèques partagées et modules
     +-mnt       Points de montage temporaires
     +-opt       Applications externes
     +-sbin      Principales commandes système
     +-tmp       Fichiers temporaires
     +-usr       Hiérarchie secondaire
     +-var       Données variables
```

Les différents systèmes de fichiers situés en dessous de la racine sont divisés en plusieurs catégories :

- Les systèmes de fichier *partageables* ou *sharables* que l'on peut utiliser entre plusieurs machines, par exemple à travers un montage NFS. Dans la liste précédente, **`/opt`** est un de ceux-là.
- Les systèmes de fichier non partageables locaux à une machine. Dans la liste précédente, **`/etc`** est un de ceux-là.
- Les systèmes de fichier *statiques* qui ne sont pas modifiés au cours du fonctionnement de la machine. Dans la liste précédente, **`/usr`** est un de ceux-là.
- Les systèmes de fichier variables qui sont modifiés au cour du fonctionnement de la machine. Dans la liste précédente, **`/var`** est un de ceux-là.

Les systèmes de fichier statiques pourront être montés en lecture seule ou *read-only*, alors que les systèmes de fichier variables devront l'être en lecture-écriture ou *read-write*. Ces derniers points peuvent avoir une grande importance dans le cas d'un système embarqué car cela peut conditionner la configuration matérielle du système au niveau du type de périphérique de stockage.

Le répertoire **`/bin`** contient les commandes utilisateurs les plus communes (par exemple, **`/bin/ls`**). Ces commandes seront accessibles à tous les utilisateurs y compris bien entendu l'administrateur du système. Le répertoire **`/bin`** ne doit pas contenir de sous-répertoire. Une liste minimale de commandes disponibles est requise par le FHS.

Le répertoire **/sbin** contient les principales commandes système. Ces commandes sont théoriquement accessibles à l'administrateur *root* mais pas aux utilisateurs standards. Cependant, certaines commandes non destructrices seront accessibles à l'utilisateur standard à condition de donner le chemin d'accès complet de la commande.

Le répertoire **/boot** a été longuement cité dans la sous-section concernant le noyau Linux. Ce répertoire contient en effet les parties statiques du noyau Linux ainsi que les fichiers auxiliaires comme la carte mémoire interne du noyau ou **System.map**. Le répertoire contient également les fichiers utilisés par le chargeur LILO.

Le répertoire **/dev** contient les pseudo-fichiers ou *devices* associés aux pilotes de périphériques. Les pilotes sont en effet accessibles à travers des fichiers spéciaux appelés également nœuds (ou *nodes*). Ces fichiers sont caractérisés par deux valeurs numériques :

- le majeur ou *major* qui identifie le pilote ;
- le mineur ou *minor* qui représente une sous-adresse en cas de présence de plusieurs périphériques identiques, contrôlés par un même pilote.

L'exemple ci-après donne la liste des fichiers spéciaux associés aux pilotes des ports séries :

```
# ls -l /dev/ttyS*
crw-------    1 root     tty        4,  64 mai  5  1998 /dev/ttyS0
crw-------    1 root     tty        4,  65 mai  5  1998 /dev/ttyS1
crw-------    1 root     tty        4,  66 mai  5  1998 /dev/ttyS2
crw-------    1 root     tty        4,  67 mai  5  1998 /dev/ttyS3
```

Dans ce cas- là, le majeur vaut 4 et les mineurs vont de 64 à 67. Pour créer une nouvelle entrée dans le répertoire **/dev**, on utilise la commande **mknod** :

```
mknod /dev/monpilote c majeur mineur
```

Le caractère « c » indique que l'on dialogue avec le périphérique en mode caractère. Ce mode indique que l'on peut échanger avec le périphérique un nombre variable d'octets. On peut aussi utiliser des périphériques en mode bloc, ce qui impose de dialoguer par blocs de données. La mémoire de masse est un exemple de périphérique en mode bloc. Dans ce cas, l'entrée dans le répertoire **/dev** sera créée avec l'option « b » à la place de « c » :

```
mknod /dev/monpilote b majeur mineur
```

Le répertoire **/etc** contient la majorité des fichiers et sous-répertoires de configuration du système. Ce répertoire ne doit pas contenir de fichiers exécutables. Les sous-répertoires sont classés en fonction du type de configuration associé, par exemple :

- **/etc/sysconfig** pour la configuration générale du système,
- **/etc/X11** pour la configuration de l'interface graphique X *Window System*.

Plus généralement, le répertoire **/etc/machin_truc** contiendra les fichiers de configurations spécifiques à l'application *machin_truc*.

Ce répertoire renferme en particulier le fichier **/etc/inittab** contenant lui-même le nom du script de démarrage lancé par le processus **/sbin/init** ou son remplaçant, lui-même lancé par le noyau Linux.

On notera également que le répertoire **/etc/rc.d** contient les scripts de démarrage du système. Linux utilise pour cela le système des niveaux d'exécution ou *run levels* introduit par Unix *System V*. Basé sur le lancement de services en fonction du niveau d'exécution, ce système a le gros avantage de définir proprement la localisation des différents scripts de démarrage dans des sous-répertoires correspondant aux niveaux :

```
# ls -l /etc/rc.d
total 60
drwxr-xr-x    2 root     root         4096 oct 23 00:27 init.d
-rwxr-xr-x    1 root     root         2889 nov  8  1999 rc
-rwxr-xr-x    1 root     root         1940 oct 15 09:30 rc.local
-rwxr-xr-x    1 root     root        13679 fév 23  2000 rc.sysinit
drwxr-xr-x    2 root     root         4096 oct 23 00:27 rc0.d
drwxr-xr-x    2 root     root         4096 oct 23 00:27 rc1.d
drwxr-xr-x    2 root     root         4096 oct 23 00:27 rc2.d
drwxr-xr-x    2 root     root         4096 oct 23 00:27 rc3.d
drwxr-xr-x    2 root     root         4096 oct 23 00:27 rc4.d
drwxr-xr-x    2 root     root         4096 oct 23 00:27 rc5.d
drwxr-xr-x    2 root     root         4096 oct 23 00:27 rc6.d
```

Les niveaux d'exécution sont les suivants :

- 0 Arrêt
- 1 Mode utilisateur unique ou *single user mode*
- 2 Multi-utilisateur (*multi-user*) sans NFS
- 3 Multi-utilisateur complet
- 4 Non utilisé
- 5 X Window System, interface graphique
- 6 Redémarrage

Ce principe de fonctionnement sera explicité plus longuement au chapitre 5 et nous verrons qu'il n'est pas forcément adapté aux contraintes d'un environnement réduit.

Le répertoire **/lib** contient les bibliothèques principales du système ainsi que les modules du noyau. Comme tous les systèmes d'exploitation modernes, Linux utilise un système de bibliothèques partagées ou *shared libraries* qui permet de ne stocker une bibliothèque utilisée par plusieurs programmes qu'une seule fois dans le système.

D'autres avantages sont induits, par exemple la mise à jour unique par simple remplacement de la bibliothèque partagée et bien sûr l'optimisation de la mémoire en cas d'utilisation simultanée de la bibliothèque par plusieurs programmes.

Le répertoire **/lib** doit contenir les bibliothèques partagées utilisées par les commandes des répertoires **/bin** et **/sbin**.

Pour connaître les bibliothèques partagées utilisées par un programme, on peut utiliser la commande **ldd** :

```
# ldd /bin/cp
        libc.so.6 => /lib/libc.so.6 (0x4001c000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

La totalité des programmes Linux utilise ces deux bibliothèques :

- **libc.so.6** est la bibliothèque principale du système aussi appelée *glibc* ;
- **ld-linux.so.2** est le chargeur ou loader qui permet de charger les bibliothèques nécessaires au programme.

Le nom d'une bibliothèque partagée correspond souvent à un lien symbolique sur la version réelle de la bibliothèque, ce qui permet de faire coexister plusieurs versions de bibliothèques partagées :

```
# ls -l /lib/libc*
-rwxr-xr-x    1 root    root    4101324 fév 29  2000 /lib/libc-2.1.3.so
lrwxrwxrwx    1 root    root         13 sep 25  2000 /lib/libc.so.6 -> libc-2.1.3.so
```

D'autres répertoires comme **/usr/lib** peuvent contenir des bibliothèques partagées utilisées par les commandes du répertoire ***/usr/bin***. Hormis ***/lib***, la liste des répertoires explorés pour la recherche des bibliothèques partagées se trouve dans le fichier **/etc/ld.so.conf** :

```
# cat /etc/ld.so.conf
/usr/X11R6/lib
/usr/lib
/usr/kerberos/lib
/usr/i486-linux-libc5/lib
/usr/local/lib
```

Si l'on souhaite ajouter un répertoire, il faut l'ajouter à cette liste et utiliser la commande **/sbin/ldconfig** pour valider la modification et créer le fichier **/etc/ld.so.cache**.

Le répertoire **/usr** est une hiérarchie secondaire c'est-à-dire qu'elle contient des sous-répertoires de commandes utilisateur comme **/usr/bin**, des commandes système comme **/usr/sbin** ou de bibliothèques partagées comme **/usr/lib**. Cette hiérarchie pourra également contenir d'autres sous-répertoires comme l'arborescence *X Window System* sur **/usr/X11R6**.

Le répertoire **/mnt** contient les points de montage temporaires comme **/mnt/cdrom**. Ces points de montage pourront être ajoutés dans le fichier ***/etc/fstab*** :

```
/dev/cdrom    /mnt/cdrom         iso9660 noauto,owner,ro 0 0
```

chaque champ indiquant respectivement le périphérique à monter, le point de montage, le type de système de fichier à utiliser et les options de montage.

Le répertoire **/tmp** est une zone de stockage de fichiers temporaires utilisés par exemple par le compilateur *gcc*. Dans le cas d'un système embarqué, on veillera à ce que ce répertoire

soit vidé automatiquement à chaque démarrage du système, ce qui n'est pas le cas des distributions standards.

Le répertoire `/var` est une zone de stockage de données variables, en particulier :

- Les fichiers verrous ou *lock files* permettant d'assurer l'unicité d'utilisation de certaines ressources comme les ports séries. Ces fichiers sont localisés sur le répertoire `/var/lock` et contiennent le numéro de processus ou PID (*Process IDentifier*) du processus ayant verrouillé le périphérique.
- Les fichiers de trace ou *log files* qui contiennent les traces d'exécution de certains programmes. Le répertoire utilisé est `/var/log` et le fichier principal de trace est `/var/log/messages`. Dans le cas d'un système embarqué, on veillera bien sûr à limiter les traces au strict minimum. La plupart des distributions utilisent cependant des systèmes de purge automatique des fichiers de trace.
- Les files d'attente ou *spool directories* qui permettent de stocker temporairement des fichiers en attente de traitement par un autre processus. Le répertoire utilisé est `/var/spool`.
- Les fichiers de fonctionnement ou *pid files* qui indiquent simplement qu'un programme donné tourne à l'instant présent avec le PID inscrit dans le fichier. Le format du fichier est le plus souvent *nom_du_programme.pid*.

Ce répertoire est un facteur de risque important concernant les problèmes de remplissage de disque et devra être traité de manière très attentive dans le cas du développement d'un système embarqué.

En résumé

Le système Linux a une structure similaire à celles des autres versions d'Unix. L'organisation des fichiers dans le système est conforme au *Filesystem Hierarchy Standard* (*FHS*) ou *Standard d'organisation du système de fichiers.*

Le noyau Linux est monolithique mais on peut charger et décharger dynamiquement des modules.

Le système de fichier virtuel `/proc` permet d'accéder facilement aux paramètres du système et particulièrement à ceux du noyau.

La compilation du noyau, et ce afin de l'adapter aux besoins de l'application, est presque toujours nécessaire dans le cas d'un projet de système embarqué.

Le chargement initial du noyau dans la mémoire du système nécessite le plus souvent l'utilisation d'un logiciel annexe appelé chargeur. LILO est un exemple fréquent de logiciel chargeur. Le noyau peut également être chargé à partir d'une disquette de démarrage.

5

Construction du système

Ce chapitre décrit la méthode générale de création d'un système Linux embarqué à partir de composants directement extraits d'une distribution classique, ou bien générés à partir des sources.

Les distributions classiques

La distribution classique vise en premier lieu non pas l'optimisation de l'espace utilisé, pas plus que l'obtention des performances optimales, mais plutôt la facilité d'installation. Linux a longtemps souffert – et souffre toujours – des sarcasmes de ses détracteurs qui le présentent comme un système ésotérique et difficile à installer. Soumis à la pression concurrentielle et marketing, car tenus de rendre des comptes à leurs actionnaires, les éditeurs de distributions ont fortement investi dans la convivialité de la procédure d'installation, ainsi que dans l'intégration optimale de la distribution dans un environnement matériel générique.

Le plus avancé sur ce domaine est l'éditeur français Mandriva (*http::/www.mandriva.com*) qui s'est clairement positionné sur ce terrain, même s'il est certainement le plus miné, car subissant le plus l'hégémonie de Microsoft. Le résultat est cependant à la hauteur des attentes comme on peut le constater sur la figure 5-1 ci-après.

Bien que le sujet de la convivialité du bureau ne soit pas notre préoccupation principale dans cet ouvrage, il faut reconnaître que Mandriva a certainement participé de la popularité de Linux parmi les utilisateurs de Windows, autrefois rebutés par le côté austère de Linux. Même si l'on peut émettre des réserves sur le fait qu'une interface graphique soit systématiquement plus simple à utiliser qu'un outil en mode texte, il faut louer Mandriva pour son effort de prosélytisme et de vulgarisation !

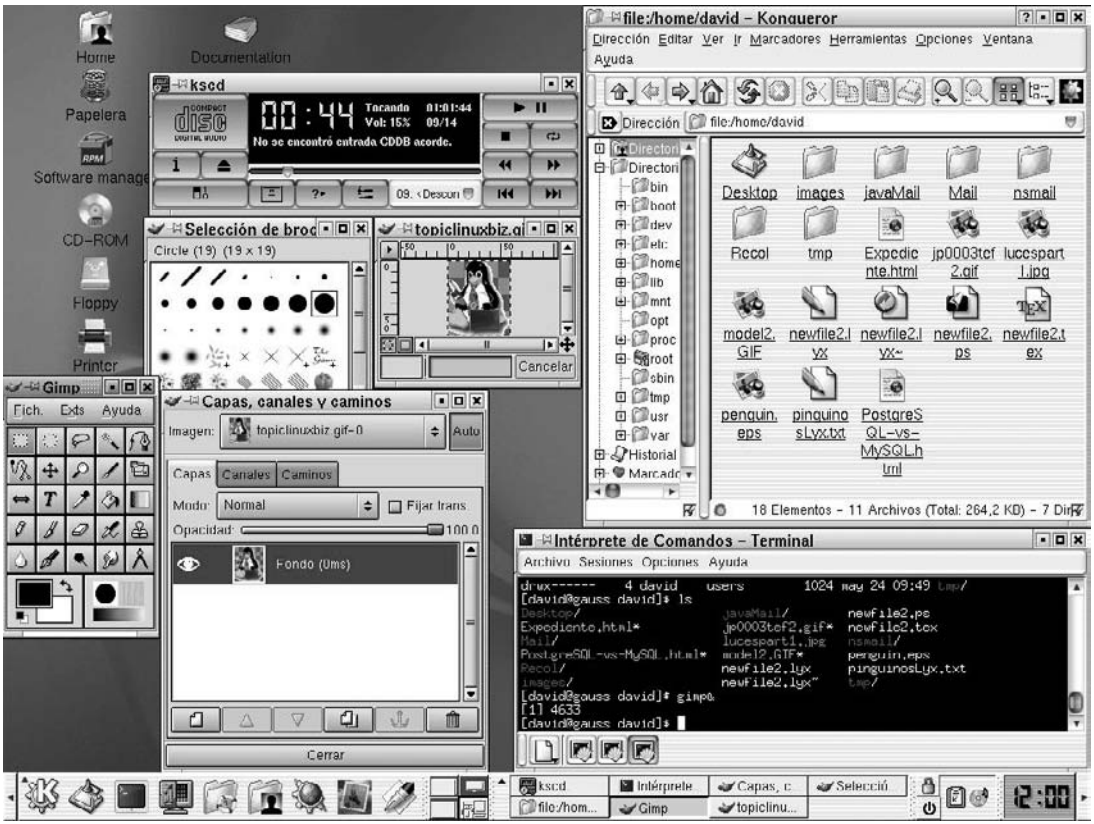


Figure 5-1

Bureau KDE multilingue de Mandriva Linux.

Le leader mondial, l'américain Red Hat Software (*http://www.redhat.com*), a toujours eu un positionnement quelque peu différent car bien plus orienté vers le cœur du système, en l'occurrence le noyau Linux. Red Hat argue du point de vue commercial que 70 % des développeurs principaux du noyau sont salariés de Red Hat et que de ce fait la société est la plus grosse concentration d'expertise Linux au monde. Force est de constater que l'argument est valide d'autant que Red Hat contrôle également la société Cygnus, figure emblématique de l'Open Source durant l'ère pré-linuxienne et employeur des principaux développeurs du compilateur gcc et du débogueur gdb. En raison du ciblage différent de la population d'utilisateurs, l'approche de la distribution Red Hat est beaucoup plus stricte comme le montre la figure 5-2 ci-après.

Pour les puristes, Red Hat fournit également une interface d'installation en mode texte.

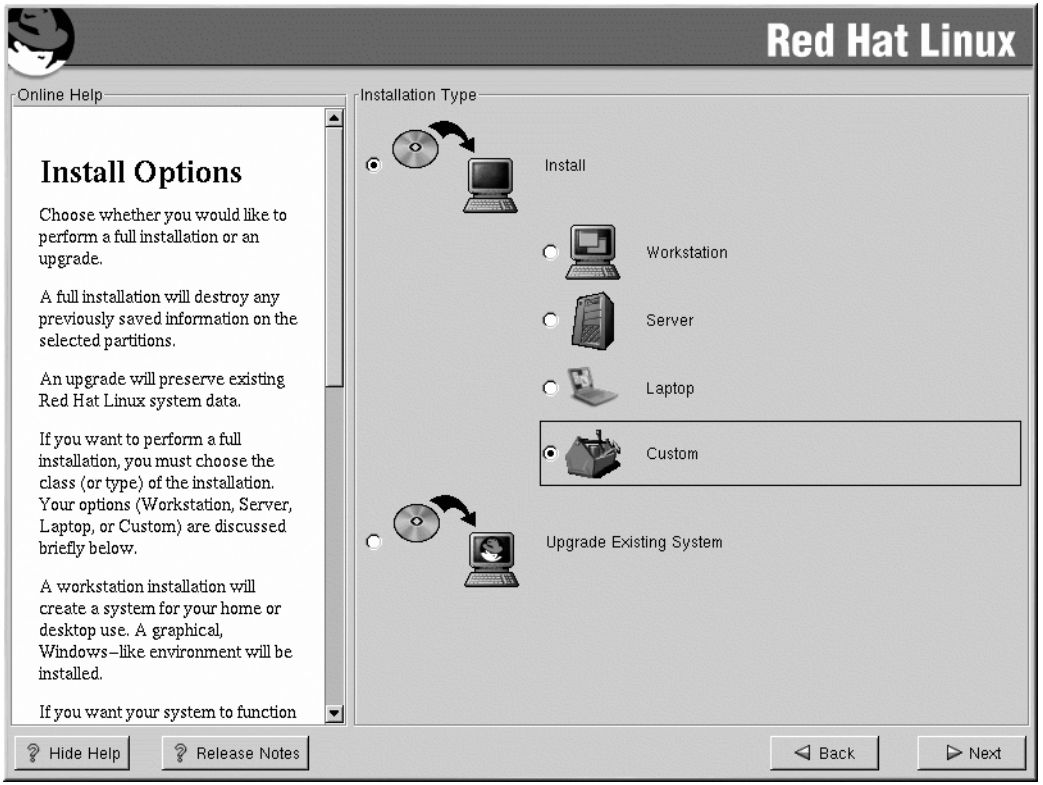


Figure 5-2

Procédure d'installation Red Hat 7.2.

Figure 5-3

Procédure d'installation Red Hat 7.2 (mode texte).

En dehors du niveau de sophistication graphique, les deux produits installent à peu près les même composants, et la procédure se passe la plupart du temps sans encombre et nécessite le moins de dialogue possible avec la machine. Cette fluidité a forcément un revers, en l'occurrence, la difficulté aujourd'hui d'installer une distribution récente dans moins d'1 Go de disque, ce qui n'est pas forcément un problème pour la cible d'utilisateur concernée puisque la taille minimale des disques était de 10 Go fin 2002 au moment de l'écriture de la première version de cet ouvrage, elle est actuellement de 40 Go. L'évaluation rapide est édifiante, sur un système Red Hat 7.2 configuré en installation « ordinateur portable ». Le volume utilisé, ne serait-ce que pour les modules du noyau, est déjà significatif :

```
cd /lib/modules/2.4.7-10/
[pierre@localhost 2.4.7-10]$ du -s .
25744   .
```

soit environ 25 Mo, sans parler de celui des bibliothèques partagées « essentielles » qui avoisine les 10 Mo rien que sur le répertoire `/lib`. Sur une distribution plus récente (Red Hat 9.0) le volume est encore de 10 Mo supérieur :

```
# cd /lib/modules/2.4.20-8/
[root@localhost 2.4.20-8]# du -s .
30296   .
```

Le volume total de ces deux seuls répertoires est déjà largement supérieur à l'espace total alloué pour la majorité des systèmes embarqués typiques, ce dernier se situant en général entre 5 et 10 Mo pour un système fonctionnel incluant les services réseau habituels tels que FTP, Telnet, HTTP ou SSH. C'est donc plus qu'un euphémisme de dire que les procédures d'installation classiques ne sont pas du tout adaptées au monde de l'embarqué.

Méthodologie générale

Au vu des conclusions auxquelles on parvient à la section précédente, quelles solutions s'offrent donc à nous ?

L'approche la plus simple est de se rabattre vers les distributions embarquées dites « spécialisées ». Celles-ci se présentent sous forme de produits commerciaux (Montavista Linux, LynuxWorks, BlueCat, etc.) ou bien de projets Open Source (LEAF, LRP, ELDK, Crosstools, etc.). Certains de ces produits fournissent des procédures d'installation conviviales et permettent au novice d'obtenir assez rapidement un système fonctionnel. Le principal problème est souvent le manque de « granularité » dans le choix des composants, ce qui a pour effet d'obtenir un résultat qui n'atteint pas le maximum d'optimisation, en particulier au niveau de la taille occupée. Cette approche est intéressante si l'on souhaite peu s'investir intellectuellement sur le sujet.

L'approche radicalement opposée consiste à partir de composants indépendants d'une quelconque distribution, par exemple LinuxFromScratch *(http://www.linuxfromscratch.org)*, puis d'assembler les briques jusqu'à obtention d'un système « parfait ». Cette approche est relativement extrémiste et nous conseillons de la limiter à quelques éléments du système.

L'approche intermédiaire est celle que nous préconiserons dans ce chapitre. Le but est de partir d'une distribution éprouvée (en l'occurrence la Red Hat 7.2), ce qui garantit la validité des différents composants.

Remarque

Les concepts exposés restent bien entendu valides dans le cas des versions plus récentes comme la Red Hat 9.0 ou les nouvelles Fedora Core 2 et 3. En cas de différences notables, celles-ci seront explicitées.

Un assemblage harmonieux de ces composants – quitte à les adapter à notre besoin – permettra d'obtenir assez rapidement un résultat fonctionnel. Cette approche a en commun avec l'approche précédente la volonté d'avoir la maîtrise et la compréhension du système, point de vue que nous défendrons systématiquement dans cet ouvrage, et qui est absolument fondamental dans l'environnement industriel et embarqué.

La méthodologie, directement issue du bon sens (qui est souvent la meilleure des méthodologies), se résume en trois points.

Le premier consiste à *assimiler* le fonctionnement du système Linux, ce qui n'est pas très compliqué puisque Linux, et plus généralement Unix, est un des systèmes les plus simples et logiques qui soient.

Le deuxième point consiste à extraire les éléments essentiels du système de manière à n'utiliser que ces derniers pour la construction de la cible finale. Dans ce domaine, la moindre hésitation concernant l'utilité d'un composant peut conduire à l'installation de données superflues si ce composant est lié à d'autres éléments.

Enfin le troisième point consiste à utiliser, en plus de la nôtre, l'intelligence des autres – en l'occurrence celle des créateurs de distributions – de manière à adapter les différents composants en fonction de l'analyse du deuxième point.

Quels sont les éléments importants d'un système Linux dans un environnement embarqué ?

À la section « Répertoires et fichiers principaux » du chapitre précédent, on a soigneusement décrit les différents composants du FHS (*Filesystem Hierarchy Standard*). Nous nous contenterons ici de lister les composants essentiels.

Sur le plan pratique, la suite du chapitre indiquera la démarche qu'il faut suivre dans le cas d'une distribution embarquée construite à partir d'une Red Hat 7.2 sur un système de type x86. La raison principale de ce choix tient à ce que sa mise en application est largement facilitée pour la majorité des lecteurs.

Le programme de démarrage

Ce composant est essentiel dans la majorité des cas car il permet de charger l'image du noyau dans la mémoire du système. Décrit au chapitre précédent, LILO, est encore le programme le plus utilisé. Certains soucis de compatibilité avec les disques de grande taille ont conduit à son remplacement par GRUB – issu du projet GNU-Hurd – dans certaines distributions comme la Red Hat version 7.2 ou supérieure.

Pour certaines architectures et cartes mère x86, il est cependant possible de remplacer le BIOS par un noyau Linux adapté, comme décrit dans le projet *http://www.linuxbios.org*.

On pourra aussi s'affranchir de l'utilisation d'un programme de démarrage si l'on crée un périphérique « bootable » – par exemple une disquette – en effectuant une copie bloc à bloc du noyau sur ce périphérique *via* les commandes **dd** et **rdev**.

Le noyau

Le noyau Linux est le seul élément réellement essentiel. Après chargement et détection des périphériques principaux, le comportement par défaut tient au démarrage du programme **/sbin/init** qui se charge de la suite du démarrage du système. Le programme de démarrage LILO comporte une option permettant de remplacer l'exécution du programme **init** par un autre, ce qui peut permettre de lancer un programme applicatif directement depuis le noyau.

```
LILO: linux init=/bin/mon_programme
```

Cette fonction est cependant relativement limitée puisque l'accès aux ressources du système devra se faire entièrement par l'applicatif en question.

L'optimisation du noyau, c'est-à-dire la sélection exclusive des fonctionnalités et pilotes nécessaires, aura un fort impact sur le temps de démarrage du système qui est souvent une contrainte primordiale pour un système embarqué.

Les fichiers de configuration (/etc)

Les fichiers de configuration sont localisés dans le répertoire **/etc**. Dans un système Linux classique, le nombre de fichiers et sous-répertoires est assez conséquent même si l'espace utilisé reste modeste. Un rapide test sur une Red Hat 7.2 donne les résultats suivants :

```
[root@localhost dev]# cd /etc
[root@localhost etc]# find . -print |  wc -l
   1437
[root@localhost etc]# du -s .
6916 .
```

L'optimisation du répertoire aura pour but initial la simplification de la structure.

Les pseudo-fichiers ou nœuds (/dev)

Les nœuds ou *nodes* sont situés dans le répertoire **/dev**. Ce ne sont pas des fichiers en tant que tels et chaque entrée ne consomme qu'un *inode* dans le système de fichiers :

```
[root@localhost etc]# du -s /dev
272  /dev
```

Les programmes essentiels (/sbin et /bin)

Les commandes essentielles à la configuration du système sont en général localisées dans le répertoire **/sbin**. Le meilleur exemple en est la commande **init** qui correspond au premier processus lancé par le noyau. Ce répertoire ne contient normalement pas de commande directement accessible à un utilisateur non privilégié.

Certaines commandes essentielles sont également situées sur `/bin`, comme la commande `mount` qui permet le montage du système de fichier principal (*root filesystem*) au démarrage. Pour simplifier la structure du système cible, toutes les commandes seront situées sur ces deux répertoires et nous n'utiliserons donc pas les répertoires `/usr/bin` et `/usr/sbin`.

Remarque

Si le système ne nécessite pas la version complète des utilitaires Linux, on pourra avantageusement se tourner vers BusyBox, (*http://www.busybox.net*), un projet Open Source très actif depuis quelques années et dont le but est de remplacer l'arborescence Linux par une version simplifiée basée sur un exécutable unique qui remplace les commandes classiques.

Dans cette nouvelle édition de l'ouvrage, nous consacrerons une partie de ce chapitre à la description de la mise en place de BusyBox.

Les bibliothèques essentielles (/lib)

Ce répertoire contient les bibliothèques partagées indispensables au fonctionnement du système, donc utilisées par les programmes situés sur `/bin` et `/sbin`.

Les répertoires variables (/var)

Le répertoire `/var` contient des fichiers et sous-répertoires dont l'encombrement peut augmenter fortement au cours du fonctionnement d'un système Linux classique. C'est la raison pour laquelle ce répertoire est habituellement situé sur une partition dédiée. Dans le cas d'un système embarqué, l'utilisation du répertoire sera limitée aux sujets suivants :

- fichiers de fonctionnement ou pid-files,
- fichiers verrous ou lock-files,
- fichiers de trace ou log-files.

Les deux premiers sont très peu consommateurs en espace. La partie fichier de trace doit être étudiée avec soin. Avec le répertoire `/tmp`, c'est le seul répertoire dont la taille est susceptible d'augmenter de manière significative durant la vie du système.

Création d'une partition dédiée

Lors de la création du système, une méthode simple est de travailler sur une partition dédiée, créée spécialement sur le poste de développement. Le remplissage de la partition se fera au fur et à mesure de l'avancement en copiant peu à peu les composants de la distribution standard – soit le disque de développement – vers la partition dédiée. Dans l'exemple qui suit, cette partition sera initialisée en utilisant le format *ext2* ou *ext3* standards sur la distribution Red Hat 7.2. Le chapitre 7 traitera plus en détail le choix final du système de fichier pour l'optimisation de la distribution définitive.

Remarque

Le format ext3 est totalement compatible avec le format ext2 sauf qu'il utilise en plus un *journal interne* des modifications qui permet une plus grande sécurité de fonctionnement et un redémarrage rapide en cas d'arrêt intempestif.

Une validation de la partition de test se fera de temps à autre en redémarrant le système sur cette partition. La taille de la partition est à évaluer selon les besoins du système cible. Lorsque le système sera validé sur le poste de développement, il restera à le transférer sur la cible et à finaliser la procédure de boot sur cette même cible.

Si la mémoire de masse utilisée par la cible est adaptable sur le poste de développement, on pourra bien sûr travailler directement sur l'espace cible. Le chapitre 3, traitant des solutions matérielles, décrit quelques exemples d'outillages permettant d'adapter des mémoires flash de type CompactFlash, DiskOnChip ou disque flash 2,5 pouces sur un PC classique.

La partition est créée avec l'outil `fdisk`. Nous utilisons un deuxième disque connecté en « esclave » sur le premier bus IDE. Ce disque est donc identifié par le device `/dev/hdb`.

```
# fdisk /dev/hdb

Commande (m pour aide) : p

Disque /dev/hdb : 15 têtes, 56 secteurs, 989 cylindres
Unités = cylindres sur 840 * 512 octets

Périphérique Amorce    Début       Fin    Blocs   Id  Système
/dev/hdb1                 1        79     33152   83  Linux
/dev/hdb2                80       158     33180   83  Linux
```

Nous créons une nouvelle partition primaire (troisième partition) d'environ 40 Mo. Nous rappelons qu'il est possible de créer 4 partitions primaires sur un disque.

```
Commande (m pour aide) : n
Action de commande
   e   Etendue
   p   Partition primaire (1-4)
p
Nombre de partitions (1-4): 3
Premier cylindre (159-989, 159 par défaut) :
Utilisation de la valeur par défaut 159
Dernier cylindre ou +size ou +sizeM ou +sizeK (159-989, 989 par défaut) : +40M
```

Une fois la partition créée, nous validons la modification.

```
Commande (m pour aide) : p
Disque /dev/hdb : 15 têtes, 56 secteurs, 989 cylindres
Unités = cylindres sur 840 * 512 octets
Périphérique Amorce    Début       Fin    Blocs   Id  Système
/dev/hdb1                 1        79     33152   83  Linux
/dev/hdb2                80       158     33180   83  Linux
/dev/hdb3               159       256     41160   83  Linux
```

```
Commande (m pour aide) : w
La table de partition a été modifiée !

Appel de ioctl() pour relire la table de partition.

AVERTISSEMENT: Si vous avez créé ou modifié
une partition DOS 6.x, reportez-vous au manuel de fdisk
pour plus d'informations.
Synchronisation des disques.
```

Il faut maintenant formater la partition en *ext3*. Pour cela, il suffit d'utiliser la commande **`mke2fs`**. L'option **`-j`** permet de créer le journal interne, donc un système *ext3*.

```
# mke2fs -j /dev/hdb3
mke2fs 1.23, 15-Aug-2001 for EXT2 FS 0.5b, 95/08/09
warning: 199 blocks unused.

Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
10320 inodes, 40961 blocks
2058 blocks (5.02%) reserved for the super user
First data block=1
5 block groups
8192 blocks per group, 8192 fragments per group
2064 inodes per group
Superblock backups stored on blocks:
        8193, 24577

Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 23 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
```

Un système de fichier de type ext2 est systématiquement vérifié à intervalles réguliers. Cette option n'est pas utile dans le cas du système de fichier ext3 et l'on pourra inhiber cette vérification en utilisant la commande **`tune2fs`**.

```
# tune2fs -i 0 /dev/hdb3
tune2fs 1.23, 15-Aug-2001 for EXT2 FS 0.5b, 95/08/09
Setting interval between check 0 seconds

# tune2fs -c 0 /dev/hdb3
tune2fs 1.23, 15-Aug-2001 for EXT2 FS 0.5b, 95/08/09
Setting maximal mount count to -1
```

La partition peut maintenant être montée sur un point de montage temporaire du type **/mnt/emb**.

```
mkdir -p /mnt/emb
mount -t ext3 /dev/hdb3 /mnt/emb
```

On peut également ajouter le montage dans la table **/etc/fstab**.

```
/dev/hdb3               /mnt/emb      ext3    noauto  0 0
```

Il suffira alors de taper **mount /mnt/emb** pour monter le système de fichier.

Création des répertoires

Le système de fichier nouvellement créé ne contient rien mis à part le répertoire **lost+found**. Ce répertoire est utilisé par le pilote du système de fichier pour stocker les portions de fichiers récupérées en cas de problème grave sur la partition.

```
# mount /mnt/emb
# cd /mnt/emb
# ls -l
total 12
drwxr-xr-x    2 root     root        12288 jui 19  2001 lost+found
```

Le fonctionnement du système Linux nécessite la création d'un certain nombre de répertoires, indispensables ou utiles.

```
cd /mnt/emb
mkdir bin boot dev etc lib proc root sbin tmp usr var
```

Le répertoire **/tmp** nécessite des droits d'accès particuliers, en l'occurrence les droits de lecture/écriture/parcours pour tous les utilisateurs.

```
chmod a+rwx /tmp
```

Pour préparer la suite de l'installation, nous pouvons créer quelques sous-répertoires utiles pour la suite.

```
mkdir -p usr/lib/kbd/keytables
```

pour stocker les tables de définition de la géométrie des claviers.

```
mkdir -p var/log var/run
```

pour stocker les éventuels fichiers de trace et de fonctionnement.

```
mkdir -p etc/sysconfig
```

pour stocker la configuration du système, en particulier au niveau réseau.

L'option **-p** indique de créer toute l'arborescence des répertoires si celle-ci est inexistante.

Le répertoire /extra

Il est intéressant de créer un répertoire contenant des commandes et bibliothèques associées, dont on se sert uniquement pour la phase de mise au point. Dans ces commandes de mise au point, on peut citer un éditeur de texte comme **vi** ou bien des commandes comme **ldd**. Le principe est de créer un répertoire **/extra** ainsi que les répertoires nécessaires comme **/extra/bin** ou **/extra/lib**. La localisation de ces composants présente cet intérêt que l'on peut à tout moment estimer l'espace qu'ils occupent, et donc avoir une bonne visibilité de l'empreinte mémoire du système. Pour obtenir un système optimisé, il suffira de supprimer le répertoire **/extra**.

La création des répertoires se fait de manière classique.

```
cd /mnt/emb
mkdir -p /extra/bin /extra/lib
```

Remarque

Il est nécessaire de modifier le fichier `/etc/ld.so.conf` afin d'y ajouter le répertoire `/extra/lib`. Il faut ensuite taper `ldconfig` pour valider cette modification. De même, pour plus de commodité, il est conseillé d'ajouter `/extra/bin` à la variable PATH.

Création des nœuds sur /dev

Les distributions Linux fournissent la commande **MAKEDEV** afin d'initialiser le répertoire **/dev** en créant sur ce dernier les différents points d'entrée ou *nœuds*. Curieusement, la commande **MAKEDEV** est située sur le répertoire **/dev** de la distribution Red Hat 7.2.

```
# rpm -ql MAKEDEV | more
/dev/MAKEDEV
/etc/makedev.d
/etc/makedev.d/00macros
/etc/makedev.d/ataraid
...
```

Pour créer les nœuds principaux sur le répertoire /**mnt/emb/dev**, il suffit de procéder à :

```
# /dev/MAKEDEV -v -d /mnt/emb/dev generic console
create mem                          c   1   1 root:kmem 640
create kmem                         c   1   2 root:kmem 640
create port                         c   1   4 root:kmem 640
...
create parport6                     c  99   6 root:lp 660
create parport7                     c  99   7 root:lp 660
```

Une rapide vérification du répertoire indique que les nœuds ont bien été créés et ne consomment que très peu d'espace.

```
# ls -l /mnt/emb/dev | wc -l
   3041
```

```
# du -s /mnt/emb/dev/
50      /mnt/emb/dev
```

On pourra bien sûr ajouter ultérieurement des nœuds en utilisant la commande **mknod** décrite au chapitre précédent.

Remplissage de /bin /et /sbin

Dans un premier temps, nous copierons seulement sur ces répertoires les programmes indispensables au démarrage d'un système minimal :

- **/sbin/init**, premier programme démarré par le noyau et donc premier processus du système (son *PID* vaut 1) ;
- **/sbin/update**, chargé de vider le cache sur le disque à intervalles réguliers ;
- **/bin/mount**, chargé de monter le système de fichier principal ou *root filesystem* ;
- **/bin/rm**, utilisé par le script de démarrage **rc.S** pour effacer certains fichiers de fonctionnement ;
- **/bin/sh**, interpréteur de commande, lancé en fin d'initialisation du système. Dans notre exemple, nous utiliserons la version classique **bash** mais il existe des versions plus légères comme **ash** (6 fois plus petit en taille) qui sont souvent plus adaptées à des distributions finales de systèmes Linux embarqués.

Il suffit pour cela de copier les fichiers au moyen des commandes :

```
cp /bin/bash /bin/mount /mnt/emb/bin
cp /sbin/init /sbin/update /mnt/emb/sbin
```

Il est également nécessaire de positionner un lien symbolique entre **bash** et **sh**.

```
cd /mnt/emb/bin
ln -s bash sh
```

Un peu d'histoire

Le programme interpréteur de commande sous Unix est appelé Bourne-Shell, en l'honneur de son créateur Steve Bourne, et ce depuis le début du développement d'Unix. La version GNU du « shell » fut donc appelée *bash*, pour Bourne-Again-Shell.

Création des bibliothèques sur /lib

Les commandes précédentes utilisent des bibliothèques partagées et le contenu du répertoire /lib est donc indissociable des répertoires **/bin** et **/sbin**. Comme précisé au chapitre 4, nous rappelons qu'il est possible de connaître la liste des bibliothèques utilisées par un programme en utilisant le script **/sbin/ldd**. L'exemple suivant indique les bibliothèques utilisées par l'interpréteur de commande **bash**.

```
# cd /mnt/emb/bin
# ldd bash
libtermcap.so.2 => /lib/libtermcap.so.2 (0x4002d000)
libdl.so.2 => /lib/libdl.so.2 (0x40031000)
libc.so.6 => /lib/i686/libc.so.6 (0x40035000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Il suffit alors de copier les bonnes bibliothèques dans le répertoire **/mnt/emb/lib**.

Cette méthode est cependant très fastidieuse et il est préférable d'utiliser pour cela le script **mklibs.sh**. Issu du projet DEBIAN *(http://www.debian.org)*, ce petit script très pratique (seulement 800 lignes de script, largement commentées) permet d'analyser une liste de fichiers exécutables, d'extraire les dépendances par rapport aux bibliothèques partagées, puis de copier automatiquement ces bibliothèques sur le répertoire de votre choix. Dans notre cas, il suffit de faire :

```
# cd /mnt/emb
# mklibs.sh -v -d /mnt/emb/lib bin/* sbin/*
Initializing data objects... done.
Constructing dependency graph... (LALNLALA) done.
Eliminating cycles... () done.
No pic archive for library libdl-2.2.4.so found, falling back to simple copy.
No pic archive for library libtermcap.so.2.0.8 found, falling back to simple copy.
No pic archive for library libc-2.2.4.so found, falling back to simple copy.
```

Miraculeusement, les bibliothèques nécessaires apparaissent dans le répertoire /**mnt/emb/lib** :

```
# ls -l /mnt/emb/lib/
total 1376
-rwxr-xr-x    1 root     root        88188 mar 22 07:11 ld-2.2.4.so
lrwxrwxrwx    1 root     root           11 mar 22 07:11 ld-linux.so.2 -> ld-2.2.4.so
-rw-r--r--    1 root     root      1282588 mar 22 07:11 libc-2.2.4.so
lrwxrwxrwx    1 root     root           13 mar 22 07:11 libc.so.6 -> libc-2.2.4.so
-rw-r--r--    1 root     root         9828 mar 22 07:11 libdl-2.2.4.so
lrwxrwxrwx    1 root     root           14 mar 22 07:11 libdl.so.2 -> libdl-2.2.4.so
lrwxrwxrwx    1 root     root           19 mar 22 07:11 libtermcap.so.2 ->
ibtermcap.so.2.0.8
-rw-r--r--    1 root     root        11832 mar 22 07:11 libtermcap.so.2.0.8
```

Lorsque vous ajouterez de nouveaux exécutables, il suffira de lancer de nouveau le script afin de mettre à jour les dépendances. Le script est disponible sur le site de la distribution DEBIAN à l'adresse *http://cvs.debian.org/boot-floppies/scripts/rootdisk*.

Dans le cas des nouvelles versions de la glibc (2.3 et supérieure), il est préférable d'utiliser la nouvelle version de `mklibs` qui n'est plus écrite en langage de script shell mais en langage Python. Cette nouvelle version est disponible à l'adresse *http://packages/debian.org/unstable/divel/interlibs.html*. L'utilisation du programme est très similaire :

```
# mklibs -v -d ./lib bin/* sbin/*
I: Using ld-linux.so.2 as dynamic linker.
I: library reduction pass 1
Objects: adduser
```

```
Object: bin/adduser
323 symbols, 323 unresolved
reducing libm.so.6
...
Moving libm.so.6 to libm.so.6.
Moving libcrypt.so.1 to libcrypt.so.1.
Moving libc.so.6 to libc.so.6.
Moving ld-linux.so.2 to ld-linux.so.2.
#
```

Ce qui conduit au remplissage du répertoire **lib** comme suit :

```
# ls -l lib
total 1496
-rwxr-xr-x    1 root     root        84936 avr  6 08:47 ld-linux.so.2
-rw-r--r--    1 root     root        18588 avr  6 08:47 libcrypt.so.1
-rw-r--r--    1 root     root      1272092 avr  6 08:47 libc.so.6
-rw-r--r--    1 root     root       136004 avr  6 08:47 libm.so.6
```

Remplissage du répertoire /etc

La simplicité étant un facteur important pour un système embarqué, nous veillerons à ce que toute la configuration du système soit localisée dans ce répertoire ou un de ses sous-répertoires. Outre la clarté de la structure, cette méthode a pour avantage de permettre la duplication rapide de la configuration d'un système en copiant quelques fichiers sur un support quelconque. Cette fonctionnalité peut être très avantageuse pour la phase d'industrialisation, de duplication de système ou bien de maintenance.

La version minimale du répertoire **/etc** contient les fichiers suivants :

```
# ls -l /mnt/emb/etc
total 16
-rw-r--r--    1 root     root           158 mar 22 07:39 fstab
-rw-r--r--    1 root     root           798 mar 22 07:39 inittab
drwxr-xr-x    2 root     root          4096 mar 22 07:39 rc.d
-rw-r--r--    1 root     root          1868 mar 22 07:39 termcap
```

Le fichier **inittab** contient la configuration du programme **/sbin/init** ; dans notre cas, il est réduit au strict minimum.

```
#
# inittab       This file describes how the INIT process should set up
#               the system in a certain run-level.
#
# Default runlevel.
id:S:initdefault:

# System initialization (runs when system boots).
si:S:sysinit:/etc/rc.d/rc.S

# End of /etc/inittab
```

Le point d'entrée *sysinit* donne le chemin d'accès au script `rc.S` d'initialisation du système.

Remarque

Le lecteur notera la simplification drastique du système de démarrage par rapport à la procédure des « niveaux d'exécution » décrite au chapitre précédent. Le fait est qu'un système embarqué démarre en général un petit nombre de services et que la complexité de la procédure standard n'est pas forcément la plus adaptée.

Dans la suite de l'évolution de notre système, le fichier `inittab` sera enrichi de l'utilisation classique du programme `getty` associé à un système d'authentification par nom d'utilisateur (ou *login*) et un mot de passe.

Le fichier `rc.S` est également réduit au strict minimum :

```
#!/bin/sh
#
# /etc/rc.d/rc.S:  System initialization script.
#
PATH=/sbin:/bin

# Start update.
/sbin/update &

# Remount the root filesystem in read-write mode
echo "Remounting root device with read-write enabled."
/bin/mount -w -n -o remount /
if [ $? -gt 0 ]; then
  echo
  echo "Attempt to remount root device as read-write failed !"
  echo "This is going to cause serious problems..."
fi
```

Le fichier `rc.S` doit également effacer le fichier `/etc/mtab` qui contient la liste des systèmes de fichiers montés.

On peut ensuite monter les systèmes listés dans le fichier `/etc/fstab`.

```
echo "Cleaning mtab."
/bin/rm -f /etc/mtab*

# Mounting local fs from /etc/fstab
echo "Mounting local filesystems."
/bin/mount -at nonfs
```

Le fichier `fstab` contient la liste des montages du système :

```
/dev/hdb3       /               ext3    defaults        1  1
none            /proc           proc    defaults        0  0
```

Ces montages sont réduits au strict minimum, en l'occurrence la partition principale et le pseudo-système de fichier `/proc` décrit au chapitre 4.

Le script `rc.S` se termine pour l'instant par le démarrage de l'interpréteur de commande.

```
# Shell
HOME=/root
USER=root
export HOME USER
cd $HOME
/bin/sh
```

Le fichier **termcap** contient la description des capacités des terminaux utilisables sur le système. Cette version réduite comprend uniquement la description de la console locale *linux*, ainsi que celle de l'éternel *vt100*. Ce fichier est utilisé par la **libtermcap**, elle-même utilisée par **bash**.

Rappel

Linux utilise le système termcap (ou terminfo) dérivé d'Unix. Ce système à la fois simple et intelligent permet d'utiliser n'importe quel type de terminal – en mode texte – sans modifier les applications, et ce en définissant simplement les capacités (*capabilities*) du terminal dans la configuration du système.

Création d'un noyau adapté

Comme nous l'avons vu au début de ce chapitre, les noyaux livrés avec les distributions standards sont largement surdimensionnés pour une utilisation en embarqué. Rappelons que les modules livrés avec le noyau standard de la Red Hat 7.2 occupent environ 25 Mo, 30 Mo dans les dernières versions, sans compter la partie statique qui occupe environ 800 Ko en version compressée. Le noyau que nous utiliserons occupera environ 600 Ko pour la partie statique et seulement quelques dizaines de kilo-octets pour les modules, si ceux-ci sont utilisés.

La compilation d'un noyau Linux a été longuement expliquée au chapitre 4. Concernant les sources à utiliser, le choix suivant s'offre à nous :

- utiliser les sources du noyau à partir du package RPM (.src.rpm) fourni sur la distribution Red Hat ;
- utiliser les sources officiels du noyau à partir de la version standard disponible sur le site *ftp.kernel.org*.

Dans le cas présent, nous aurons tendance à recourir à la seconde solution, principalement parce que le noyau officiel offre systématiquement la dernière version à jour, et ce pour toutes les architectures.

Le principal travail d'optimisation du noyau consistera en la compréhension des diverses options de ce dernier au travers des procédures **make menuconfig** ou **make xconfig** décrites précédemment. Ensuite, l'on s'attachera à élaguer les options valides pour ne conserver que celles spécifiquement utilisées pour notre système. Les principales options décrites sont celles dont l'utilisation peut générer un surcoût de taille ou de temps de démarrage du système.

Au moment de l'écriture de la mise à jour de cet ouvrage se pose le problème du choix entre le noyau 2.4 et le noyau 2.6. Le saut technologique n'est cependant pas si grand que cela et une distribution conçue pour le noyau 2.4 pourra être facilement adaptée au noyau 2.6 sous réserve de mise à jour de quelques composants comme le paquetage `modutils` déjà cité précédemment. Si l'on regarde le coté technique interne du noyau 2.6, les différences sont cependant notables et nous citerons :

- l'intégration de la fonctionnalité de noyau préemptif que nous expliciterons au chapitre 9 ;
- une meilleure intégration avec le projet µClinux permettant le support des processeurs dans MMU ;
- le support matériel encore amélioré en particulier sur les périphériques récents comme l'USB ou les nouveaux périphériques audio via ALSA ;
- le modèle de périphérique unifié et l'utilisation du système de fichiers /sys (de type sysfs) permettant de visualiser l'arbre des périphériques contrôlés par le noyau ;
- l'amélioration du support multithread avec l'utilisation de NPTL (pour *Native POSIX Thread Library*) ;
- la fonction d'hyperthreading permettant de gérer certains processeurs (comme les P4) de manière virtuelle.

Des documents très complets concernant les nouvelles fonctionnalités du noyau 2.6 sont bien entendu disponibles sur Internet. Nous pouvons entre autres citer l'article « Wonderful world of Linux 2.6 » par Joseph Pranevich (*jpranevich@kniggit.net*) disponible à l'adresse *http://www.kniggit.net/wwol26.html*. Une traduction française de l'article est disponible à l'adresse *http://dsoulayrol.free.fr/articles/wonderful_2.6.html*.

En résumé, nous pouvons dire que face au choix entre le noyau 2.4 et le noyau 2.6, l'utilisateur pourra avoir l'attitude suivante :

- La migration vers 2.6 ne s'impose pas lorsqu'un système existant fonctionne correctement et que l'on n'ajoute pas de nouveaux périphériques ou protocoles qui nécessiteraient un portage régressif (ou backport) vers le noyau 2.4.
- Dans le cas d'un nouveau projet, l'utilisation du 2.6 est conseillée car certains projets comme RTAI (couche temps réel dur pour Linux étudiée au chapitre 9) utilisent le noyau 2.6 comme version de référence. Il conviendra cependant de vérifier que le support des périphériques matériels sur 2.6 est au même niveau que sur 2.4 mais la tendance va dans ce sens.

Le support des modules

Le support correspond à l'entrée *Loadable module support*. Dans le cas d'un système de petite taille, on peut envisager de ne pas utiliser de modules et donc de limiter le noyau à sa composante statique. Ce choix peut être motivé par des contraintes de sécurité, en particulier pour empêcher l'ajout dynamique de fonctionnalités au noyau. La suppression

des modules permet également de simplifier la structure du système de par l'absence du répertoire `/lib/modules`.

Dans la majorité des cas, il est cependant recommandé de valider le support des modules – ainsi que celui de `kmod` – afin de conserver la compatibilité fonctionnelle avec les systèmes Linux classiques, et aussi de se réserver la possibilité de mise à jour d'un pilote sans modifier la partie statique du noyau. Si le support des modules est conservé, il est recommandé de valider dans la mesure du possible les options du noyau sous forme de modules, et ce en cochant l'option *M* au lieu de *Y*. Certaines fonctions comme le support de disque dur ou le système de fichier principal devront cependant être validées dans la partie statique du noyau.

On pourra également ajouter les commandes système correspondant à la manipulation des modules dans l'espace utilisateur même si cela n'est pas indispensable au niveau de la cible.

```
[root@localhost linux-2.4]# ls -l /sbin/insmod
-rwxr-xr-x    1 root     root        90476 oct  2 18:09 /sbin/insmod
[root@localhost linux-2.4]# ls -l /sbin/lsmod
lrwxrwxrwx    1 root     root            6 fév  7 14:05 /sbin/lsmod -> insmod
[root@localhost linux-2.4]# ls -l /sbin/rmmod
lrwxrwxrwx    1 root     root            6 fév  7 14:05 /sbin/rmmod -> insmod
[root@localhost linux-2.4]# ls -l /sbin/modprobe
lrwxrwxrwx    1 root     root            6 fév  7 14:05 /sbin/modprobe -> insmod
[root@localhost linux-2.4]# ls -l /sbin/depmod
-rwxr-xr-x    1 root     root        54444 oct  2 18:09 /sbin/depmod
```

Le type de processeur

Le choix du type de processeur a habituellement assez peu d'importance dans le cas de systèmes classiques, même si cela peut influer sur les performances de la machine. Le cas des systèmes embarqués est différent car ils sont souvent basés sur des processeurs moins puissants (compatibles Pentium ou 486), et il est donc indispensable de spécifier le type de processeur sous peine de ne pouvoir exécuter le noyau. Les processeurs AMD de type Duron devront aussi être spécifiés de la même manière.

Attention

Certaines architectures anciennes ne fonctionnent pas – ou mal – avec un noyau 2.4, et ce bien qu'elles soient annoncées compatibles 486. Il est donc important de vérifier la compatibilité du noyau Linux auprès du fournisseur de processeur ou bien de carte. On pourra le cas échéant se rabattre sur un noyau plus ancien (2.2) mais cela peut entraîner des problèmes d'évolutivité, en particulier concernant la disponibilité de pilotes de périphériques.

Les périphériques en mode bloc

Cette dénomination concerne tout ce qui correspond à la mémoire de masse (disquette, disques durs, interfaces SCSI ou autres interfaces spécifiques). Les entrées correspondantes dans le menu de configuration du noyau sont *Block devices*, *Memory Technology Devices (MTD)*, *ATA/IDE/MFM/RLL* et *SCSI*.

Le menu Block devices correspond à la validation de divers périphériques spéciaux comme ceux pilotés par le port parallèle. La plupart de ces options devront être invalidées dans la majorité des cas. Notez cependant que la fonction de disque mémoire (RAM disk) est validée par ce menu. Si vous désirez utiliser les techniques de RAM disk décrites plus avant dans l'ouvrage, vous devrez obligatoirement valider cette option. La validation du support des disquettes est également très utile car elle permettra de démarrer le noyau ou bien de faire des transferts de fichiers, par exemple en utilisant la commande **`tar`** directement sur le nom de device associé à la disquette.

```
tar xvf /dev/fd0
```

Il faut également noter que le pilote de périphérique fourni par le constructeur M-Systems pour les flashes DiskOnChip est constitué d'un « patch » du noyau qui ajoutera l'entrée suivant dans le menu *Block devices* :

Figure 5-4

Configuration du support DiskOnChip.

L'entrée *Loopback device support* permet de configurer un device spécial très utilisé dans le monde Linux. Il correspond aux entrées **`/dev/loopx`** (x variant de 0 à 9) et permet de monter une image d'un système de fichier exactement comme l'on monterait le système de fichiers lui-même. Par exemple, si vous disposez d'un fichier .iso correspondant à une image à graver sur CD-Rom, on pourra utiliser la commande :

```
mount -t iso9660 -o loop mon_image.iso /mnt/cdrom
```

Cette fonctionnalité sera également très utile pour les images *CRAMFS* citées plus bas.

Le menu MTD concerne les mémoires flash ne disposant pas d'interfaces IDE. Notons que le noyau Linux supporte également la majorité des périphériques DiskOnChip à travers l'interface MTD, ce qui peut avantageusement remplacer le pilote fourni par M-Systems. Ce menu permet également de valider l'utilisation du système de fichier JFFS2 (*Jounalling Flash Filesystem, version 2*) dont nous parlerons plus bas.

Attention

Le pilote MTD ne fonctionne pas pour l'instant sur la dernière génération de flash « Millenium Plus » développée par M-Systems.

Le menu ATA/IDE/MFM/RLL est très souvent utilisé car il concerne les périphériques IDE qui représentent aujourd'hui l'interface disque la plus utilisée, particulièrement dans le monde x86. Comme nous l'avons dit dans le chapitre 3 concernant les choix matériels, l'utilisation de périphériques IDE est très avantageuse car elle évite l'ajout de pilotes spéciaux et permet de manipuler plus facilement ces périphériques dans le monde des PC

classiques. Ce choix a tout de même quelques inconvénients, en particulier au niveau du coût mais aussi au niveau du temps de démarrage du système car la détection des périphériques sur le bus IDE prendra souvent plus de temps que pour une flash non-IDE. Le support des disques IDE devra donc être validé, et ce dans la partie statique du noyau, afin de permettre l'accès au disque au moment du boot. La validation du support des différents *chipsets* devra être adaptée en fonction de votre matériel.

La partie SCSI est le plus souvent invalidée car les systèmes de petite taille utilisent rarement ce type d'interface.

La configuration réseau

D'après les statistiques décrites au chapitre 2, 16 % des utilisateurs choisissent Linux pour son excellent support réseau, ce qui représente à peine moins que le critère de disponibilité des sources (20 %), l'absence de coût de licence (19 %) ou la robustesse (18 %). Tout cela pour dire que la majorité des applications embarquées sous Linux utiliseront le réseau et que sa configuration est donc un point sensible. La configuration est divisée en trois parties :

- les protocoles ;
- les options de fonctionnement ;
- les adaptateurs matériels (cartes).

Les deux premiers points sont accessibles au travers du menu *Networking options*. La configuration des cartes est accessible par le menu *Network devices*. Une configuration embarquée prendra soin d'inhiber le support de tous les protocoles non utilisés (AppleTalk, IPX, DECnet). Pour le protocole TCP/IP, on pourra invalider les options non utilisées comme l'*aliasing*, le support de *firewalling*, le *multicast* ou le *tunnelling*. En revanche, l'on n'oubliera pas de valider le support du protocole PPP (*Point to Point Protocol*) si l'équipement doit être connecté à un réseau de type IP *via* une connexion modem ou ADSL.

Le choix de l'adaptateur réseau se fera également en fonction des architectures matérielles possibles. Si les cartes (ou la carte) sont validées dans la partie statique, elles seront automatiquement détectées au démarrage du noyau et ce avant l'initialisation du réseau. Si le support est validé sous forme de module, alors ce dernier sera chargé par `kmod` au moment de l'initialisation du réseau à condition que la configuration soit indiquée dans le fichier `/etc/modules.conf` pour 2.4 ou `/etc/modprobe.conf` pour 2.6.

```
alias eth0 eepro100
```

Les systèmes de fichiers

Comme nous l'avons dit précédemment, le système de fichiers constitue un périphérique virtuel qui sera piloté de la même manière qu'un périphérique matériel. On parle alors de *pilote* et donc de support de tel ou tel système de fichiers. Les distributions classiques

valident le support pour les formats communément utilisés sur les PC classiques en plus des systèmes de fichiers natifs de Linux. En l'occurrence, les formats FAT, VFAT (Windows) et ISO9660 (CD-Rom) seront systématiquement disponibles.

Dans le cas d'un système embarqué, on pourra valider uniquement le support ext3 dans le cas où l'on utilise un disque muni d'une flash de type IDE, ou bien ne rien valider du tout si l'on utilise une flash non-IDE directement pilotée par la couche MTD. Le système de fichiers pourra alors être de type JFFS2. Le menu contient également la validation du système de fichier CRAMFS (*Compressed RAM File System*), permettant de créer une image compressée d'un répertoire. L'image pourra ensuite être montée et utilisée avec les restrictions que nous expliciterons plus tard.

Les périphériques en mode caractère

Ce type de périphérique est le plus fréquent. Le type « mode caractère » indique que l'on peut dialoguer avec le périphérique au niveau de l'octet contrairement aux périphériques en mode bloc avec lesquels on n'échange que des blocs de données, par exemple 2048 octets pour les lecteurs de CD-Rom.

Ces périphériques incluent :

- les ports séries standards ou cartes dites « multi-voies »,
- les ports parallèles,
- les périphériques de pointage (souris ou équivalent),
- les pseudo-terminaux de type System V (Unix98 ptys),
- les consoles virtuelles Linux,
- le support du bus *I2C*.

Définitions

Linux utilise les pseudo-terminaux de type BSD (Berkeley Software Distribution) qui correspondent aux entrées `ttyxx` et `ptyxx` du répertoire /dev. Les versions récentes de Linux qui utilisent la glibc-2.1 permettent cependant de se servir des pseudo-terminaux hérités de System V. Lorsqu'un processus désire obtenir un pseudo-terminal, il accède au nœud `/dev/ptmx` et le pseudo-terminal alloué sera disponible en tant que `/dev/pts/numéro`, équivalent au `/dev/ttyp2` sous BSD. Il sera donc nécessaire de créer le répertoire `/dev/pts` ainsi que l'entrée `/dev/ptmx` au moyen de la commande `mknod /dev/ptmx c 5 2`.

Ce menu contient également la validation du support de la console Linux sur le port série :

Figure 5-5

Support de console sur le port série.

Par défaut, la console sera dirigée sur l'écran local du système (la sortie VGA pour un PC classique). Dans un système embarqué, cette sortie n'est pas toujours disponible et il est alors très intéressant de rediriger la console sur un terminal asynchrone de type émulateur de terminal. La configuration d'une telle console est très simple. Outre la validation de l'option susmentionnée, il est nécessaire de spécifier les paramètres de la console au moment du démarrage à travers les options passées au noyau par LILO. La syntaxe à utiliser est du type :

```
LILO: linux console=ttyS0
```

En plus du port série utilisé – ici **/dev/ttyS0** qui correspond au port COM1 du PC –, on peut spécifier la vitesse, la parité et le nombre de bits de données.

```
LILO: linux console=ttyS0,19200n8
```

La valeur par défaut est 9600 bits/s, pas de parité, et 8 bits de données.

L'ajout systématique d'une telle ligne se fera grâce à la commande **append** de LILO dans le fichier **/etc/lilo.conf** décrit au chapitre précédent.

```
append="console=ttyS0"
```

Cette ligne sera ajoutée soit au niveau global (avant la première commande **image**), soit de manière spécifique sur une image donnée. Une documentation complète sur la configuration d'une console sur port série est disponible dans le fichier **Documentation/serial-console.txt** dans les sources du noyau Linux.

Génération du noyau

Lorsque les choix sont validés, la configuration du noyau est sauvegardée en activant l'option *Save and Exit* du menu principal. La génération du noyau se fait alors par un classique :

```
make dep; make clean; make bzImage; make modules
```

Si vous avez validé le support des modules, il faudra installer ceux-ci grâce à la commande :

```
make modules_install
```

qui installera l'arborescence **/lib/modules/2.4.x** sur votre système de développement (*x* correspondant au niveau de patch du noyau). Le répertoire des modules devra ensuite être copié dans l'arborescence du système embarqué – soit **/mnt/emb -**, ainsi que le fichier **/etc/modules.conf**.

Remarque

La copie devra être faite en respectant les droits et attributs des fichiers initiaux. On peut pour cela utiliser les options `-a` et `-r` de la commande `cp` :

```
cp -a -r source destination
```

ou bien utiliser une combinaison des commandes `find` et `cpio`.

```
cd source
find . -print | cpio -pdv source
```

Test du système

Le premier test du système peut s'effectuer en copiant l'image du noyau sur une disquette de démarrage.

```
cd /usr/src/linux-2.4
dd < arch/i386/boot/bzImage > /dev/fd0
rdev /dev/fd0 /dev/hdb3
rdev -R /dev/fd0 1
```

Si l'on considère que le système de fichier principal correspond à **`/dev/hdb3`**, lorsqu'on redémarre la machine en utilisant cette disquette de boot on doit obtenir :

```
Linux version 2.4.18 (root@duron) (gcc version 2.96 20000731 (Red Hat Linux 7.1
2.96-98)) #3 mar mar 26 18:18:08 CET 2002
BIOS-provided physical RAM map:
 BIOS-e820: 0000000000000000 - 000000000009fc00 (usable)
 BIOS-e820: 000000000009fc00 - 00000000000a0000 (reserved)
 BIOS-e820: 00000000000f0000 - 0000000000100000 (reserved)
 BIOS-e820: 0000000000100000 - 0000000007ff0000 (usable)
 BIOS-e820: 0000000007ff0000 - 0000000007ff8000 (ACPI data)
 BIOS-e820: 0000000007ff8000 - 0000000008000000 (ACPI NVS)
 BIOS-e820: 00000000fec00000 - 00000000fec01000 (reserved)
 BIOS-e820: 00000000fee00000 - 00000000fee01000 (reserved)
 BIOS-e820: 00000000ffee0000 - 00000000fff00000 (reserved)
 BIOS-e820: 00000000fffc0000 - 0000000100000000 (reserved)
On node 0 totalpages: 32752
zone(0): 4096 pages.
zone(1): 28656 pages.
zone(2): 0 page.
Kernel command line: auto BOOT_IMAGE=2_4_18_emb_zb ro root=343 BOOT_FILE=/boot/b
zImage-2.4.18_emb_zb console=ttyS0
Initializing CPU#0
Detected 896.198 MHz processor.
Console: colour VGA+ 80x25
Calibrating delay loop... 1789.13 BogoMIPS
Memory: 127016k/131008k available (919k kernel code, 3604k reserved, 225k data,
188k init, 0k highmem)
Dentry-cache hash table entries: 16384 (order: 5, 131072 bytes)
Inode-cache hash table entries: 8192 (order: 4, 65536 bytes)
Mount-cache hash table entries: 2048 (order: 2, 16384 bytes)
Buffer-cache hash table entries: 4096 (order: 2, 16384 bytes)
Page-cache hash table entries: 32768 (order: 5, 131072 bytes)
CPU: L1 I Cache: 64K (64 bytes/line), D cache 64K (64 bytes/line)
CPU: L2 Cache: 64K (64 bytes/line)
Intel machine check architecture supported.
Intel machine check reporting enabled on CPU#0.
CPU: AMD Duron(tm) Processor stepping 01
Enabling fast FPU save and restore... done.
Checking 'hlt' instruction... OK.
Posix conformance testing by UNIFIX
PCI: PCI BIOS revision 2.10 entry at 0xfdb01, last bus=1
```

```
PCI: Using configuration type 1
PCI: Probing PCI hardware
Unknown bridge resource 0: assuming transparent
PCI: Using IRQ router SIS [1039/0008] at 00:02.0
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
Starting kswapd
Journalled Block Device driver loaded
parport0: PC-style at 0x378 [PCSPP(,...)]
pty: 256 Unix98 ptys configured
Serial driver version 5.05c (2001-07-08) with MANY_PORTS SHARE_IRQ SERIAL_PCI enabled
ttyS00 at 0x03f8 (irq = 4) is a 16550A
ttyS01 at 0x02f8 (irq = 3) is a 16550A
lp0: using parport0 (polling).
Real Time Clock Driver v1.10e
block: 128 slots per queue, batch=32
Uniform Multi-Platform E-IDE driver Revision: 6.31
ide: Assuming 33MHz system bus speed for PIO modes; override with idebus=xx
SIS5513: IDE controller on PCI bus 00 dev 15
SIS5513: chipset revision 208
SIS5513: not 100% native mode: will probe irqs later
   ide0: BM-DMA at 0xff00-0xff07, BIOS settings: hda:DMA, hdb:DMA
   ide1: BM-DMA at 0xff08-0xff0f, BIOS settings: hdc:DMA, hdd:DMA
hda: WDC WD400BB-00CLB0, ATA DISK drive
hdb: WDC AC2420H, ATA DISK drive
hdc: CRD-8240B, ATAPI CD/DVD-ROM drive
ide0 at 0x1f0-0x1f7,0x3f6 on irq 14
ide1 at 0x170-0x177,0x376 on irq 15
hda: 78165360 sectors (40021 MB) w/2048KiB Cache, CHS=4865/255/63
hdb: 830760 sectors (425 MB) w/128KiB Cache, CHS=989/15/56
hdc: ATAPI 24X CD-ROM drive, 128kB Cache
Uniform CD-ROM driver Revision: 3.12
Partition check:
 hda: hda1 hda2 hda3
 hdb: hdb1 hdb2 hdb3
Floppy drive(s): fd0 is 1.44M
FDC 0 is a post-1991 82077
PCI: Found IRQ 11 for device 00:0b.0
3c59x: Donald Becker and others. www.scyld.com/network/vortex.html
00:0b.0: 3Com PCI 3c905B Cyclone 100baseTx at 0xd400. Vers LK1.1.16
Linux video capture interface: v1.00
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 8192 bind 8192)
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
EXT3-fs: INFO: recovery required on readonly filesystem.
EXT3-fs: write access will be enabled during recovery.
kjournald starting.  Commit interval 5 seconds
EXT3-fs: recovery complete.
```

```
EXT3-fs: mounted filesystem with ordered data mode.
VFS: Mounted root (ext3 filesystem) readonly.
Freeing unused kernel memory: 188k freed
INIT: version 2.78 booting
Remounting root device with read-write enabled.
EXT3 FS 2.4-0.9.17, 10 Jan 2002 on ide0(3,67), internal journal
Cleaning files.
Mounting local filesystems.
bash-2.05#
```

Nous disposons d'ores et déjà d'un système Linux fonctionnel et ce sur environ 2 Mo. Nous pourrons ensuite installer le chargeur de démarrage LILO comme décrit dans le chapitre 4. Pour cela, il faut tout d'abord copier le programme LILO sur la cible :

```
cp /sbin/lilo /mnt/emb/sbin
```

Si le système est installé sur un périphérique type CompactFlash IDE comme décrit au chapitre 3, celui-ci sera souvent maître du deuxième bus IDE, soit **`/dev/hdc`**. Il faudra donc configurer le fichier **`/etc/lilo.conf`** comme suit :

```
boot=/dev/hdc
read-only
vga = normal
# End LILO global section

image = /boot/bzImage
        root = /dev/hdc1
        label = linux
```

Au niveau du répertoire **`/boot`**, il faudra copier le noyau en tant que fichier **`bzImage`** ainsi que les fichiers LILO, soit **`boot.b`** et **`map`**. Une fois que le système est démarré à partir de la disquette, il suffit de taper **`lilo`** pour écrire la configuration dans le secteur de démarrage. On peut aussi écrire le secteur de démarrage depuis la partition de développement en utilisant l'option **`-r`** de **`lilo`** qui effectue un *chroot* avant de lire la configuration.

```
/sbin/lilo -r /mnt/emb -v
```

Le système est cependant relativement limité (seulement 4 commandes) et les deux chapitres suivants porteront sur l'extension de ce système, en particulier au niveau des fonctionnalités réseau, inexistantes dans la version actuelle bien que l'adaptateur réseau soit détecté.

Cas de l'utilisation de BusyBox

Le composant BusyBox permet de remplacer une bonne partie des composants Linux habituels par des versions simplifiées. L'avantage est bien évidemment un gain de place lorsque l'empreinte mémoire est limitée et que le système n'a pas besoin de toutes les fonctionnalités des commandes habituelles. De plus BusyBox est bien adapté à la compilation croisée et il simplifiera donc la mise en place d'un environnement cible dans le cas de processeurs spécifiques.

Les sources de BusyBox sont disponibles à l'adresse *http://www.busybox.net*. Une fois l'archive extraite dans un répertoire de sources, on peut paramétrer la compilation de BusyBox en utilisant une syntaxe similaire à celle du noyau Linux, soit :

```
$ make menuconfig
```

Cette commande conduit à l'affichage de la fenêtre suivante :

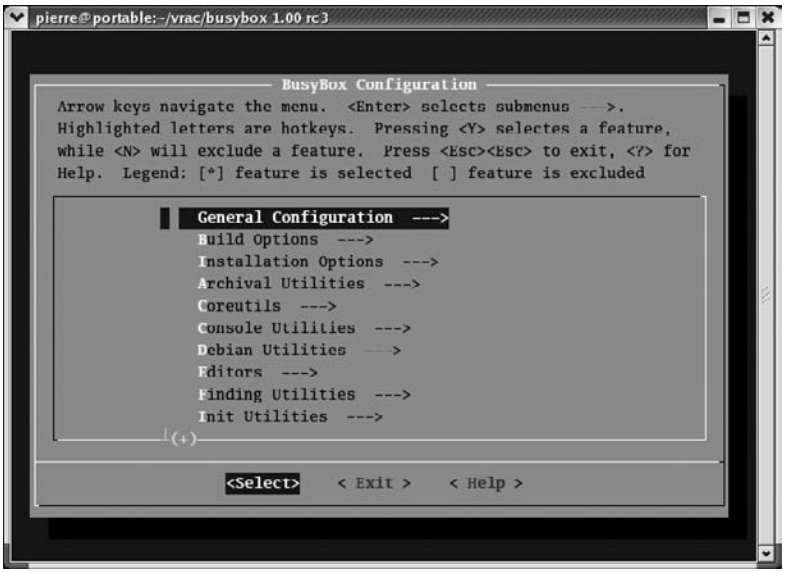


Figure 5-6

Configuration de BusyBox avant compilation.

Le nombre d'options est assez important et il n'est pas possible de les détailler ici. La documentation d'installation est assez succinte (le fichier INSTALL de la distribution) mais les menus de configuration sont clairs et il est aisé d'arriver à une configuration fonctionnelle en choisissant les options par défaut.

Après configuration de l'environnement, on peut compiler BusyBox en faisant :

```
$ make dep
$ make
```

On peut alors installer la distribution sur le répertoire cible, par exemple **`/mnt/emb`**.

```
# make PREFIX=/mnt/emb install
```

Cela a pour effet d'installer l'exécutable **`busybox`** ainsi que les autres programmes qui constituent des liens symboliques vers le fichier **`/bin/busybox`**.

```
lrwxrwxrwx    1 0      0    17 Apr  5 11:37 [ -> ../../bin/busybox
lrwxrwxrwx    1 0      0    17 Apr  5 11:37 awk -> ../../bin/busybox
```

```
lrwxrwxrwx    1 0      0   17 Apr  5 11:37 basename -> ../../bin/busybox
lrwxrwxrwx    1 0      0   17 Apr  5 11:37 bunzip2 -> ../../bin/busybox
lrwxrwxrwx    1 0      0   17 Apr  5 11:37 bzcat -> ../../bin/busybox
...
```

Dans notre cas, le répertoire **/etc** est réduit à sa plus simple expression :

```
/ # cd /etc
/etc # find . -print
.
./init.d
./init.d/rcS
./fstab
```

Le fichier **rcS** contient uniquement l'appel à la commande de montage **mount** :

```
# cat init.d/rcS
#! /bin/sh

/bin/mount -a
```

Il est bien entendu nécessaire d'utiliser *mklibs* pour remplir le répertoire **/lib** comme décrit précédemment. Dans notre cas, ce répertoire est simplifié puisque la commande **busybox** est la seule présente :

```
/ # cd /lib/
/lib # ls -l
-rwxr-xr-x    1 0        0      84936 Apr  5 11:15 ld-linux.so.2
-rw-r--r--    1 0        0    1272092 Apr  5 11:15 libc.so.6
-rw-r--r--    1 0        0      18588 Apr  5 11:15 libcrypt.so.1
-rw-r--r--    1 0        0     136004 Apr  5 11:15 libm.so.6
drwxr-xr-x    3 0        0       1024 Apr 10 12:57 modules
```

Au final, nous obtenons très facilement un système complet qui contient un grand nombre de commandes Linux, y compris des utilitaires système de configuration de réseau ou de redémarrage.

La trace qui suit décrit le démarrage d'un système basé sur BusyBox et installé sur une simple clé USB. Le paramétrage du noyau pour démarrer sur une clé USB sera décrit à la fin du chapitre 7.

```
---------------------------------
 WAITING FOR A WHILE (1000)
 TO DETECT THE USB DISK
scsi0 : SCSI emulation for USB Mass Storage devices
  Vendor: USB 2.0   Model: FLASH DISK-SX     Rev: 1.06
  Type:   Direct-Access                      ANSI SCSI revision: 02
SCSI device sda: 130656 512-byte hdwr sectors (67 MB)
sda: assuming Write Enabled
sda: assuming drive cache: write through
 sda: sda1 sda2
Attached scsi removable disk sda at scsi0, channel 0, id 0, lun 0
Attached scsi generic sg0 at scsi0, channel 0, id 0, lun 0,  type 0
```

```
---------------------------------

kjournald starting.  Commit interval 5 seconds
EXT3-fs: mounted filesystem with ordered data mode.
VFS: Mounted root (ext3 filesystem) readonly.
Freeing unused kernel memory: 188k freed

Please press Enter to activate this console.

BusyBox v1.00-rc3 (2005.04.05-10:33+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

/ #
```

En résumé

Ce chapitre fondamental nous a appris à créer un système Linux minimal mais fonctionnel à partir de composants élémentaires. Nous avons également vu que, de par la validation par défaut d'un grand nombre d'options, les distributions classiques n'étaient pas adaptées à la création directe d'un système embarqué.

Le nombre de bibliothèques partagées qu'il faut installer sur le système cible dépend directement des programmes qui y sont installés. On peut déduire la liste de ces bibliothèques en utilisant le script **`mklibs.sh`**.

De même, on peut générer la majorité des entrées du répertoire **`/dev`** en utilisant la commande **`/dev/MAKEDEV`**.

La configuration d'un noyau adapté demande certes une bonne connaissance des options dont il est doté, mais permet de réduire de manière drastique l'espace utilisé, particulièrement au niveau des modules.

Dans le cas d'un nouveau projet, l'utilisation du noyau 2.6 est préférable surtout si l'on désire utiliser des composants annexes dont le noyau de référence est souvent le 2.6.

Dans le cas d'une distribution simple, l'utilisation de BusyBox permet de simplifier grandement la mise en place d'un système.

6

Configuration du réseau

Comme nous l'avons rappelé au chapitre précédent, la majorité des systèmes Linux embarqués nécessitent des fonctionnalités réseau. Au niveau du noyau, la disponibilité du réseau implique les bons choix de configuration pour :

- les protocoles,
- les options de fonctionnement,
- les adaptateurs matériels.

Même si le réseau n'est pas fonctionnel sur le système minimal construit au chapitre 5, on peut cependant remarquer dans la trace du démarrage du système qu'une bonne partie de la détection est déjà effectuée, en particulier la détection de l'adaptateur et des protocoles supportés.

```
3c59x: Donald Becker and others. www.scyld.com/network/vortex.html
00:0b.0: 3Com PCI 3c905B Cyclone 100baseTx at 0xd400. Vers LK1.1.16
Linux video capture interface: v1.00
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 8192 bind 8192)
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
```

La présence de ces messages n'indique cependant pas un réseau fonctionnel. Pour que le réseau soit totalement activé, il est nécessaire de compléter l'initialisation de l'interface Ethernet et des tables de routage en utilisant les commandes **`ifconfig`** et **`route`**.

Dans l'exemple présenté plus haut, le support de l'adaptateur réseau (carte 3Com 3c905B) était validé dans la partie statique. En cas de support par module, la détection de l'adaptateur s'effectue au moment de l'initialisation de l'interface par **`ifconfig`**.

On peut d'ores et déjà enrichir notre système en y ajoutant ces deux commandes qui ne nécessitent pas de nouvelles bibliothèques partagées.

```
cp /sbin/ifconfig /sbin/route /mnt/emb/sbin
```

Remarque

Dans le cas de l'utilisation de BusyBox, les commandes `ifconfig` et `route` sont déjà disponibles ainsi que d'autres utilitaires réseau. La suite du chapitre et en particulier l'ajout des bibliothèques NSS citées plus loin reste cependant toujours valable.

La commande ifconfig

La commande **`ifconfig`** permet de connaître l'état d'une interface réseau et de la configurer. Utilisée en mode lecture, **`ifconfig`** retournera la liste et la configuration des interfaces du système.

```
# ifconfig

eth0      Lien encap:Ethernet  HWaddr 00:03:47:B6:FA:EA
          inet adr:192.168.3.11  Bcast:192.168.3.255  Masque:255.255.255.0
          UP BROADCAST NOTRAILERS RUNNING  MTU:1500  Metric:1
          RX packets:465 errors:0 dropped:0 overruns:0 frame:0
          TX packets:400 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0
          RX bytes:49726 (48.5 Kb)  TX bytes:167486 (163.5 Kb)

lo        Lien encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:20 errors:0 dropped:0 overruns:0 frame:0
          TX packets:20 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0
          RX bytes:1400 (1.3 Kb)  TX bytes:1400 (1.3 Kb)
```

L'interface **`lo`** ou interface *locale* est systématiquement utilisable si la couche TCP/IP du système est correctement initialisée, et ce même si le système ne dispose pas d'autre adaptateur réseau. L'adresse IP associée à l'interface locale est toujours 127.0.0.1.

Si l'on précise le nom de l'interface, **`ifconfig`** affichera uniquement la configuration de cette interface.

```
# ifconfig eth0

eth0      Lien encap:Ethernet  HWaddr 00:03:47:B6:FA:EA
          inet adr:192.168.3.11  Bcast:192.168.3.255  Masque:255.255.255.0
          UP BROADCAST NOTRAILERS RUNNING  MTU:1500  Metric:1
          RX packets:513 errors:0 dropped:0 overruns:0 frame:0
          TX packets:427 errors:0 dropped:0 overruns:0 carrier:0
          collisions:49
          RX bytes:52927 (51.6 Kb)  TX bytes:172257 (168.2 Kb)
```

Les interfaces **`eth`*x***, comme **`eth0`** ou **`eth1`**, correspondent respectivement au premier et au second adaptateur Ethernet détectés dans le système. Si une interface PPP (*Point to Point*

Protocol) est initialisée dans le système, l'interface correspondante sera `ppp0` pour la première interface, `ppp1` pour la deuxième, et ainsi de suite.

La commande `ifconfig` fait partie des commandes système essentielles et elle est donc située sur le répertoire `/sbin`. De ce fait, elle n'est pas accessible directement à l'utilisateur non privilégié, sauf si ce dernier utilise le chemin d'accès complet de la commande. Pour un utilisateur non privilégié, la commande n'est bien entendu accessible que pour des opérations de lecture de l'état des interfaces et non pour des opérations de configuration.

```
[pierre@localhost pierre]$ type ifconfig
bash: type: ifconfig: not found
[pierre@localhost pierre]$ ifconfig
bash: ifconfig: command not found
[pierre@localhost pierre]$ /sbin/ifconfig lo
lo        Lien encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0
          RX bytes:560 (560.0 b)  TX bytes:560 (560.0 b)
```

En mode configuration – donc uniquement pour le super-utilisateur *root* –, la commande `ifconfig` est utilisée le plus souvent pour l'initialisation des paramètres suivants d'une interface réseau :

- l'adresse IP,
- le masque du réseau ou *netmask*,
- le masque de diffusion ou *broadcas*t.

Les deux derniers paramètres peuvent être en général calculés automatiquement en fonction de l'adresse IP car celle-ci détermine la *classe* d'adresse utilisée par le système. Le calcul sera effectué par le script d'initialisation du réseau lors de la procédure de démarrage du système.

Rappel

Le système d'adressage IP (version 4) subdivise les adresses en sous-ensembles appelés classes. Sachant qu'une adresse IP est du type a.b.c.d, la classe est définie par la valeur a du premier élément de l'adresse. Le tableau ci-après donne la correspondance entre la valeur de a et les autres paramètres.

Tableau 6-1. Affectation des classes de réseau

Valeur de a	Classe	Masque diffusion	Masque réseau
Entre 0 et 127	A	a.255.255.255	255.0.0.0
Entre 128 et 191	B	a.b.255.255	255.255.0.0
Entre 192 et 223	C	a.b.c.255	255.255.255.0
Supérieur à 224	D (réservé)	a.b.c.255	255.255.255.0

On peut donc initialiser manuellement les interfaces **lo** et **eth0** de notre système en utilisant les commandes suivantes, pour **lo** :

```
/sbin/ifconfig lo 127.0.0.1
```

et pour **eth0** :

```
/sbin/ifconfig eth0 192.168.3.3 netmask 255.255.255.0 broadcast 192.168.3.255
```

La commande route

La commande **route** permet de manipuler les *tables de routage* IP du noyau Linux. Une table de routage décrit le chemin emprunté par un paquet IP partant du système courant et à destination d'une autre adresse ou groupe d'adresses. L'entrée dans la table de routage décrit une *interface* employée (par exemple, **eth0**) ainsi que la *passerelle* ou *gateway* à utiliser pour faire cheminer ces paquets. Comme pour la commande **ifconfig**, il est possible à l'utilisateur non privilégié d'utiliser **route** pour connaître les tables de routage du noyau. Sur une station classique connectée au réseau local, on obtient :

```
$ /sbin/route -n
Table de routage IP du noyau
Destination     Passerelle      Genmask         Indic Metric Ref    Use Iface
192.168.3.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
0.0.0.0         192.168.3.1     0.0.0.0         UG    0      0        0 eth0
```

Le résultat indique les tables de routage pour les interfaces Ethernet **eth0** et locales **lo**. La dernière ligne indique le routage *par défaut* sur la passerelle 192.168.3.1. Un résultat identique est obtenu en utilisant la commande **netstat -nr**.

On peut donc poursuivre l'initialisation de notre interface **lo** en faisant, en tant que super-utilisateur :

```
/sbin/route add -net 127.0.0.0 netmask 255.0.0.0 dev lo
```

ce qui initialise la table de routage pour les paquets transitant sur l'interface locale. De même, pour l'interface **eth0** :

```
/sbin/route add -net 192.168.3.0 netmask 255.255.255.0 dev eth0
```

Le paramètre passé à l'option **-net** indique la valeur du *réseau*. Il correspond à la valeur du masque de diffusion en remplaçant les valeurs 255 par 0. On pourrait également définir une table de routage vers une adresse donnée en utilisant une option du type **-host adresse_distante**.

L'utilisation de la commande **route -n** en lecture permet de vérifier la valeur des tables de routage. Notez que nous n'avons pas défini de passerelle, sachant que le test se limite pour l'instant à l'interface locale (**lo**) ou bien à un réseau Ethernet local (**eth0**). Pour ajouter une valeur de passerelle, on utilisera l'option **gw** de la commande **route**. En particulier, la ligne suivante indiquera que tous les paquets IP externes au réseau local devront être routés par la passerelle **192.168.3.1**.

```
/sbin/route add default gw 192.168.3.1
```

Premier test des interfaces en ICMP

On dispose d'une méthode simple pour tester le réseau en utilisant la commande **ping**, laquelle permet d'envoyer des trames ICMP (*Internet Control Message Protocol*) vers une adresse IP donnée. Pour cela, il faut ajouter la commande **ping** sur la cible embarquée. Ce programme utilise cependant une bibliothèque partagée supplémentaire qu'il faut également ajouter à la cible.

```
# ldd /bin/ping
        libresolv.so.2 => /lib/libresolv.so.2 (0x4002d000)
        libc.so.6 => /lib/i686/libc.so.6 (0x4003f000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Pour ce faire, on peut soit exécuter de nouveau le script **mklibs.sh** comme décrit dans le chapitre précédent, soit ajouter la bibliothèque à la main en exécutant :

```
cp -a /lib/libresolv* /mnt/emb/lib
```

On peut ensuite tester la connexion Ethernet sur le réseau local.

```
bash-2.05# ping 192.168.3.1
PING 192.168.3.1 (192.168.3.1) from 192.168.3.3 : 56(84) bytes of data.
64 bytes from 192.168.3.1: icmp_seq=0 ttl=255 time=648 usec
64 bytes from 192.168.3.1: icmp_seq=1 ttl=255 time=4.825 msec
64 bytes from 192.168.3.1: icmp_seq=2 ttl=255 time=356 usec
--- 192.168.3.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.356/1.943/4.825/2.041 ms
```

Si l'on ajoute une règle de routage par défaut, on peut tester l'émission de trames ICMP vers des machines de l'Internet.

```
bash-2.05# route add default gw 192.168.3.1
bash-2.05# ping 213.56.52.14
PING 213.56.52.14 (213.56.52.14) from 192.168.3.3 : 56(84) bytes of data.
64 bytes from 213.56.52.14: icmp_seq=0 ttl=255 time=363 usec
64 bytes from 213.56.52.14: icmp_seq=1 ttl=255 time=375 usec
--- 213.56.52.14 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.363/0.369/0.375/0.006 ms
```

Test de services TCP

Le test précédent ne permet pas de valider totalement l'interface car les services classiques de type http, telnet ou ftp utilisent le protocole TCP (*Transport Control Protocol*) et non ICMP. Le problème est le même pour les services basés sur UDP (*User Datagram Protocol*) qui se situent au même niveau que le protocole TCP. Si nous effectuons un test sur le service FTP – utilisant le protocole TCP sur le port numéro 21 –, nous obtenons :

```
bash-2.05# ftp
ftp: ftp/tcp: unknown service
```

La liste des services TCP et UDP et les ports associés sont en effet définis dans le fichier **/etc/services** que nous devons donc copier sur la cible. Nous copions également le fichier **/etc/protocols** qui donne la liste des protocoles IP associés à leurs numéros d'identification.

```
cp /etc/services /mnt/emb/etc
cp /etc/protocols /mnt/emb/etc
```

Hélas, cette condition nécessaire n'est pas suffisante et le service FTP n'est toujours pas fonctionnel :

```
bash-2.05# ftp
ftp: ftp/tcp: unknown service
```

La raison de ce dysfonctionnement est l'absence de configuration de la fonction NSS ou *Name Service Switch*. La NSS est une fonction de la glibc2 héritée du système Solaris de SUN Microsystems. Le but de la NSS est de pouvoir étendre de manière dynamique les fonctions réseau de la glibc2 de manière dynamique, en utilisant des bibliothèques chargées au cours de l'exécution du programme. Ces bibliothèques ne sont pas visibles avec la commande **ldd** et elles sont accessibles par les fonctions de type **dlopen**. Avant l'apparition de la NSS et de la glibc2, la libc5 devait gérer de manière statique toutes les fonctionnalités réseau ajoutées, comme la gestion des DNS (*Domain Name Service*) ou des NIS (*Network Information Service*). La NSS permet donc à des contributeurs externes d'ajouter des services sans toucher au code de la glibc2, ce dernier gardant donc une taille raisonnable, même si la glibc2 est déjà assez volumineuse.

Les différents modules gérés par la NSS sont identifiés par un nom de service dans une base de données (*networks*, *ethers*, *protocols*, *hosts*, etc.). La méthode de recherche est définie dans le fichier **/etc/nsswitch.conf**.

```
# /etc/nsswitch.conf
#
# Name Service Switch configuration file.
#

passwd:     db files nis
shadow:     files
group:      db files nis

hosts:      files nisplus nis dns
networks:   nisplus [NOTFOUND=return] files

ethers:     nisplus [NOTFOUND=return] db files
protocols:  nisplus [NOTFOUND=return] db files
rpc:        nisplus [NOTFOUND=return] db files
services:   nisplus [NOTFOUND=return] db files
```

Le mot-clé `files` indique que l'on utilisera une configuration statique définie dans un fichier de configuration, en général situé dans le répertoire **/etc**. De la même manière, **nisplus** indique que l'on recherchera l'information sur une base NIS+ si celle-ci est disponible.

Si l'on considère un service *NOM*, le code de gestion de ce dernier sera disponible dans un module **`libnss_NOM`**. Dans le cas de Linux qui gère les bibliothèques partagées, cela correspondra à une bibliothèque partagée du type **`libnss_NOM.so.2`**. Nous pouvons vérifier cela sur un système Linux complet.

```
# ls -l /lib/libnss_*
-rwxr-xr-x    1 root     root       311370 sep  4  2001 /lib/libnss_compat-2.2.4.so
lrwxrwxrwx    1 root     root           23 nov  1 17:05 /lib/libnss_compat.so.1 ->
                                                      ➥ libnss1_compat-2.2.4.so
lrwxrwxrwx    1 root     root           22 nov  1 17:05 /lib/libnss_compat.so.2 ->
                                                      ➥ libnss_compat-2.2.4.so
-rwxr-xr-x    1 root     root        72296 sep  4  2001 /lib/libnss_dns-2.2.4.so
lrwxrwxrwx    1 root     root           20 nov  1 17:05 /lib/libnss_dns.so.1 ->
                                                      ➥ libnss1_dns-2.2.4.so
lrwxrwxrwx    1 root     root           19 nov  1 17:05 /lib/libnss_dns.so.2 ->
                                                      ➥ libnss_dns-2.2.4.so
-rwxr-xr-x    1 root     root       261460 sep  4  2001 /lib/libnss_files-2.2.4.so
lrwxrwxrwx    1 root     root           22 nov  1 17:05 /lib/libnss_files.so.1 ->
                                                      ➥ libnss1_files-2.2.4.so
lrwxrwxrwx    1 root     root           21 nov  1 17:05 /lib/libnss_files.so.2 ->
                                                      ➥ libnss_files-2.2.4.so
-rwxr-xr-x    1 root     root       100440 sep  4  2001 /lib/libnss_hesiod-2.2.4.so
lrwxrwxrwx    1 root     root           22 nov  1 17:05 /lib/libnss_hesiod.so.2 ->
                                                      ➥ libnss_hesiod-2.2.4.so
-rwxr-xr-x    1 root     root       354826 aoû 31  2001 /lib/libnss_ldap-2.2.4.so
lrwxrwxrwx    1 root     root           20 nov  1 17:13 /lib/libnss_ldap.so.2 ->
                                                      ➥ libnss_ldap-2.2.4.so
-rwxr-xr-x    1 root     root       325176 sep  4  2001 /lib/libnss_nis-2.2.4.so
-rwxr-xr-x    1 root     root       355236 sep  4  2001 /lib/libnss_nisplus-2.2.4.so
lrwxrwxrwx    1 root     root           23 nov  1 17:05 /lib/libnss_nisplus.so.2 ->
                                                      ➥ libnss_nisplus-2.2.4.so
lrwxrwxrwx    1 root     root           20 nov  1 17:05 /lib/libnss_nis.so.1 ->
                                                      ➥ libnss1_nis-2.2.4.so
lrwxrwxrwx    1 root     root           19 nov  1 17:05 /lib/libnss_nis.so.2 ->
                                                      ➥ libnss_nis-2.2.4.so
```

Dans le cas où le fichier **`/etc/nsswitch.conf`** serait absent, le service *hosts* utilisera par défaut l'accès *dns*, puis l'accès *files*. Si nous copions simplement la bibliothèque **`libnss_files.so.2`**, nous avons maintenant accès au site distant par la commande **`ftp`** à condition d'utiliser les adresses IP numériques. Nous copions également le fichier **`nsswitch.conf`**.

```
cp /etc/nsswitch.conf /mnt/emb/etc
cp -a /lib/libnss_files* /mnt/emb/lib
```

Nous testons cela ensuite sur la cible.

```
# ftp
ftp> open 192.168.3.1
Connected to 192.168.3.1 (192.168.3.1).
220 ca-ol-bordeaux-9-14.abo.wanadoo.fr FTP server (Version wu-2.6.0(1) Fri Jun .
Name (192.168.3.1): pierre
```

```
331 Password required for pierre.
Password:
230 User pierre logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
```

Nous pouvons maintenant renseigner le fichier de définition du DNS **`/etc/resolv.conf`** avec une adresse de serveur DNS valide.

```
# cat /etc/resolv.conf
nameserver 62.161.120.17
```

Mais cela ne fonctionne pas :

```
# ftp ftp.kernel.org
ftp: ftp.kernel.org: Name or service not known
```

On corrige cela en copiant la bibliothèque **`libnss_dns.so.2.`**

```
cp -a /lib/libnss_dns* /mnt/emb/lib
```

On peut alors poursuivre ce test sur la cible.

```
# ftp ftp.kernel.org
Connected to ftp.kernel.org (204.152.189.113).
220 ProFTPD [ftp.kernel.org]
Name (ftp.kernel.org): anonymous
331 Anonymous login ok, send your complete email address as your password.
Password:
230-                        Welcome to the

                        LINUX KERNEL ARCHIVES
                            ftp.kernel.org
```

Conclusion

L'utilisation des bibliothèques libnss est fondamentale dans un environnement utilisant la glibc2. L'évolution de la cible vers de nouveaux services devra être systématiquement accompagnée de l'installation des bibliothèques libnss associées.

Scripts de configuration du réseau

Pour les tests précédents, nous avons effectué la configuration du réseau directement dans la ligne de commande. Cette méthode n'est bien entendu pas utilisable dans le cas d'un système embarqué réel qui devra démarrer sans aucune action d'un opérateur. Le principe utilisé ici consiste à concentrer la configuration du réseau dans un seul fichier **`network`** situé sur le répertoire **`/etc/sysconfig`**. Cette méthode a le gros avantage de simplifier la structure du système mais aussi de permettre à un utilisateur dépourvu d'expertise Linux de pouvoir configurer le réseau assez simplement. Un autre avantage, annexe, est la possibilité de dupliquer la configuration d'un système en copiant un simple fichier.

Si nous reprenons la fin du script de démarrage **/etc/rc.d/rc.S**, nous trouvons les lignes suivantes :

```
# Mount local fs from /etc/fstab
echo "Mounting local filesystems."
/bin/mount -at nonfs

# Shell
HOME=/root
USER=root
export HOME USER
cd $HOME
/bin/bash
```

L'initialisation du réseau devra se faire juste avant le lancement de l'interpréteur de commande /**bin/sh**. Dans notre cas, nous séparerons l'initialisation du réseau en deux phases :

- l'initialisation des interfaces réseau, comme effectué précédemment,
- le lancement des *services* réseau si nécessaire.

Le second point concerne les programmes serveurs qui pourraient être démarrés sur la cible, par exemple des démons FTP, HTTP, Telnet ou NTP.

Définition

Dans la terminologie Unix, un *démon* (issu de l'anglais *daemon*) est un programme lancé en tâche de fond et chargé d'assurer un service. Les programmes serveurs FTPD et HTTPD (comme Apache) sont des exemples de programmes de ce type.

La première phase d'initialisation du réseau correspond physiquement à un script **/etc/rc.d/rc.inet1**. La phase de lancement des services correspond au script **/etc/rc.d/rc.inet2**. Au niveau du script de démarrage, il suffit donc d'ajouter les lignes :

```
# Network
/etc/rc.d/rc.inet1 start
/etc/rc.d/rc.inet2 start
```

avant le lancement du shell.

Les scripts sont écrits en langage shell et nous utilisons la notion de variable d'environnement Unix pour centraliser la configuration du réseau dans **/etc/sysconfig/network**. Le fichier a l'allure suivante :

```
# Network configuration
HOSTNAME=embtest
DOMAINNAME=localdomain
DEVICE=eth0
IPADDR=192.168.3.3
GATEWAY=192.168.3.1
NETMASK=255.255.255.0
```

```
NETWORK=192.168.3.0
BROADCAST=192.168.3.0
DNS1=62.161.120.17
DNS2=
```

Au niveau du script **rc.inet1**, le contenu du fichier précédent est « évalué » au début du script.

```
#!/bin/sh
# Variables globales
. /etc/sysconfig/variables
# Réseau
. $NETCFG
```

Le fichier **variables** contient des variables d'environnement communes à tous les fichiers de configuration.

```
BASE=/etc/sysconfig
NETCFG=$BASE/network
GENCFG=$BASE/general
HOSTS=/etc/hosts
RESOLV=/etc/resolv.conf
DAEMONS=
```

Bien que le système n'utilise pas le système de niveau d'exécution (run level), le script est prévu pour démarrer les interfaces réseau, en passant le paramètre *start*, ou bien pour les arrêter en passant le paramètre *stop*. On utilise pour cela un simple test **case/esac** au niveau du script.

```
case $1 in

start)
# Démarrage du réseau

stop)
# Arrêt du réseau

*)     echo "Usage: rc.inet1 {start|stop}"
        exit 1
        ;;
esac
```

La partie démarrage du script **rc.inet1** comprend les phases suivantes.

Initialisation de l'interface locale

Cette phase reprend simplement les lignes de commandes utilisées pour les tests.

```
# Loopback
/sbin/ifconfig lo 127.0.0.1
/sbin/route add -net 127.0.0.0 netmask 255.0.0.0 dev lo
/sbin/ifconfig lo up
```

Initialisation de l'interface Ethernet

La valeur peut être obtenue soit directement dans le fichier **network** (**IPADDR**), soit en interrogeant un serveur DHCP (*Domain Host Control Protocol*). Dans notre cas, nous utilisons le client DHCP **dhcpcd** fourni sur la distribution Red Hat 7.2. Cette méthode permet de récupérer automatiquement les paramètres réseau pour la connexion Ethernet sous forme d'un fichier **/etc/dhcpc/dhcpcd-eth0.info**, si **eth0** est le nom de l'interface Ethernet.

```
IPADDR=192.168.3.4
NETMASK=255.255.255.0
NETWORK=192.168.3.0
BROADCAST=192.168.3.255
GATEWAY=192.168.3.1
DOMAIN=localdomain
DNS=62.161.120.17
DHCPSID=127.0.0.1
DHCPGIADDR=0.0.0.0
DHCPSIADDR=192.168.3.1
DHCPCHADDR=00:A0:24:78:34:6A
DHCPSHADDR=00:60:97:67:4B:79
DHCPSNAME=
LEASETIME=600
RENEWALTIME=300
REBINDTIME=525
```

Pour le fichier **/etc/sysconfig/network**, on déclare une connexion DHCP en n'initialisant aucune variable à l'exception du nom de l'interface Ethernet (**DEVICE=eth0**). On en déduit le code du script, qui attend l'écriture du fichier **dhcpcd-eth0.info**.

```
if [ "$DEVICE" != "" -a "$IPADDR" = "" ]; then
        # DHCP
        rm -rf /etc/dhcpc/* /var/run/dhcpcd-${DEVICE}.pid $RESOLV
        DHCPFILE=/etc/dhcpc/dhcpcd-${DEVICE}.info
        DHCP_OK=

        echo -n "Using DHCP for ${DEVICE}"
        /sbin/dhcpcd -n -H ${DEVICE}
        for i in 1 2 3 4 5 6 7 8 9 10 11 12
        do
                echo -n "."
                if [ -r ${DHCPFILE} ]; then
                        DHCP_OK=true
                        DOMAINNAME=$DOMAIN
                        break

                fi
                usleep 2000000
        done
```

```
        if [ "$DHCP_OK" = "" ]; then
                echo "ERROR !"
        else
                echo "OK"
                . ${DHCPFILE}
                DNS1=${DNS}
                DOMAINNAME=${DOMAIN}
        fi
```

L'initialisation de l'interface, du serveur de nom et du routage par défaut est entièrement effectuée par le démon **dhcpcd**.

Dans le cas d'une adresse IP statique, on calcule automatiquement les valeurs de réseau, masque de réseau et masque de diffusion (**NETWORK**, **NETMASK** et **BROADCAST**), si celles-ci ne sont pas définies dans le fichier **network**. Les valeurs sont calculées en respectant les règles décrites dans le tableau 6-1. L'extrait de code ci-après donne le calcul effectué en cas de classe A :

```
  # Static IP
    echo "# DNS list" > $RESOLV

    if [ "$IPADDR" != "" ]; then
        # Network, Netmask & broadcast generated from IPADDR
        set `echo ${IPADDR} | sed -e "s/\./ /g"` 0 0 0 0
        if [ $1 -ge 0 -a $1 -le 127 ]
        then
            # Class A
            if [ "$NETWORK" = "" ]; then
                NETWORK=${1}.0.0.0
            fi
            if [ "$NETMASK" = "" ]; then
                NETMASK=255.0.0.0
            fi
            if [ "$BROADCAST" = "" ]; then
                BROADCAST=${1}.255.255.255
            fi
        elif [ $1 -ge 128 -a $1 -le 191 ]
...
fi
```

Ensuite, l'interface est initialisée au moyen de la commande **ifconfig** et les règles de routage, dont le routage par défaut, sont mises en place grâce à la commande **route**.

```
 # Ethernet
/sbin/ifconfig ${DEVICE} ${IPADDR} broadcast ${BROADCAST} netmask ${NET}
/sbin/route add -net ${NETWORK} netmask ${NETMASK} ${DEVICE}
/sbin/ifconfig ${DEVICE} up

# external addresses
if [ "$GATEWAY" != "" ]
```

```
then
    /sbin/route add default gw ${GATEWAY}
fi
```

La fin de l'initialisation se résume au calcul dynamique du fichier **/etc/resolv.conf** afin d'affecter les adresses des serveurs DNS primaires et secondaires :

```
# DNS
if [ "$DNS1" != "" ]; then
    echo "nameserver $DNS1" >> $RESOLV
fi

if [ "$DNS2" != "" ]; then
    echo "nameserver $DNS2" >> $RESOLV
fi
```

Calcul du nom du système et création du fichier hosts

La fin du script initialise le nom du système grâce à la commande **hostname**. Par défaut, le nom du système est **localhost.localdomain**.

```
# Set system name
if [ "$DOMAINNAME" != "" ]; then
    DOM=".${DOMAINNAME}"
fi

if [ "$HOSTNAME" != "" ]; then
    /bin/hostname ${HOSTNAME}${DOM}
else
    /bin/hostname localhost${DOM}
fi
```

Enfin, le fichier **/etc/hosts** est créé dynamiquement à partir d'un prototype **/etc/hosts.ref** :

```
# /etc/hosts file
if [ "$DEVICE" != "" ]; then
        if [ "$DOMAINNAME" = "" ]; then
                DOMAINNAME="some.where"
        fi
        cat ${HOSTS}.ref | sed -e "s/IPADDR/${IPADDR}/g" -e "s/HOSTNAME/${HOSTNAME)"
else
        cat ${HOSTS}.ref | grep -v IPADDR > $HOSTS
fi
```

La partie arrêt (stop) du script **rc.inet1** se résume simplement aux lignes suivantes :

```
stop)
        /sbin/ifconfig lo down
        /sbin/ifconfig ${DEVICE} down
        ;;
```

Le script **rc.inet2** est beaucoup plus simple car il se borne à explorer la liste définie dans la variable **DAEMONS** :

```
#!/bin/sh

. /etc/sysconfig/variables
. $NETCFG

case $1 in
        start)
        echo -n "Starting daemons: "

        for i in $DAEMONS
        do
            if [ -x /sbin/$i ]; then
                echo -n "$i "
                /sbin/$i &
            fi
        done
        echo "";;

        stop)
        echo -n "Stopping daemons: "
        for i in $DAEMONS
        do
            if [ -x /sbin/$i ]; then
                echo -n "$i "
                /bin/killall $i
            fi
        done
        echo "";;

        *)
        echo "Usage: rc.inet2 {start|stop}"
        exit 1;;
esac
```

Si la variable **DAEMONS** n'est pas définie, le script n'aura alors aucun effet. Notez que le script suppose que les démons soient localisés sur le répertoire **/sbin**.

Au total, la taille des scripts **rc.inet1** et **rc.inet2** est inférieure à 200 lignes de code, les commentaires compris. Même si la même complexité de configuration n'est pas gérée, on peut être tenté de comparer avec les plusieurs centaines de lignes ou plus des scripts d'initialisation réseau dans une distribution classique.

Mise en place de services réseau

Sous Linux, on peut définir un service réseau de deux façons :

- En utilisant un programme démon autonome, répondant à l'adresse IP du système sur un port donné. On parle alors de programme *stand-alone*.
- En utilisant le principe du « super-démon » **inetd**, ce dernier permettant de mettre en place et d'administrer plus facilement un service réseau.

En pratique, un bon nombre de services réseau utilisent **inetd** ou **xinetd**. Les services comme Apache qui utilisent le mode *stand-alone* le font pour des raisons de performances car le passage par le super-démon réduit l'efficacité du service dans le cas d'un nombre important de connexions simultanées.

Dans le cas présent, nous allons mettre en place un service interne à **xinetd**, en l'occurrence le service *rdate*, défini dans le document RFC-868, et qui permet d'obtenir à distance la date d'un système.

```
# rdate 192.168.3.3
[192.168.3.3]   Mon Apr  8 19:05:15 2002
```

Nous devons tout d'abord installer le super-démon **xinetd**, ainsi que les bibliothèques et fichiers de configuration associés. Le programme **xinetd** utilise quelques bibliothèques partagées supplémentaires que nous devrons copier dans le répertoire /**lib**.

```
# ldd /usr/sbin/xinetd
        libnsl.so.1 => /lib/libnsl.so.1 (0x4002d000)
        libm.so.6 => /lib/i686/libm.so.6 (0x40043000)
        libcrypt.so.1 => /lib/libcrypt.so.1 (0x40066000)
        libc.so.6 => /lib/i686/libc.so.6 (0x40093000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Soit **libnsl**, **libcrypt** et **libm**.

Il faut ensuite copier le fichier de configuration **/etc/xinetd.conf** et certains fichiers du répertoire **/etc/xinetd.d** en fonction des services que l'on veut installer. Chaque service est défini par un fichier de configuration dans **xinetd.d**.

```
# ls -l /etc/xinetd.d
total 64
-rw-r--r--    1 root     root          295 mar  3 23:59 chargen
-rw-r--r--    1 root     root          295 mar  3 23:59 daytime
-rw-r--r--    1 root     root          287 mar  3 23:59 echo
-rw-r--r--    1 root     root          306 mar  3 23:59 echo-udp
-rw-r--r--    1 root     root          343 mar  3 23:59 linuxconf-web
-rw-r--r--    1 root     root          317 mar  3 23:59 rsync
-rw-r--r--    1 root     root          406 mar  3 23:59 sgi_fam
-rw-r--r--    1 root     root          302 mar  3 23:59 telnet
-rw-r--r--    1 root     root          318 mar  3 23:59 time
-rw-r--r--    1 root     root          314 mar  3 23:59 time-udp
```

Les fichiers à copier pour **rdate** sont **time** et **time-udp**.

Le programme **xinetd** nécessite également la présence d'une liste d'utilisateurs **/etc/passwd** car il fait référence à l'utilisateur *root* dans ses fichiers de configuration. La mise en place de l'authentification des utilisateurs sera explicitée au chapitre suivant mais, pour l'instant, nous définirons le fichier **/etc/passwd** suivant :

```
root::0:0:Super User:/root:/bin/bash
```

Il suffit pour finir de déclarer le nom du service dans **/etc/sysconfig/network** afin que le script **/etc/rc.d/rc.inet2** le prenne en compte au prochain démarrage.

```
# Daemons
DAEMONS="xinetd"
```

Au démarrage suivant, nous obtenons la trace du lancement de **xinetd**.

```
Starting daemons: xinetd
```

Et nous pouvons donc tester le service depuis un autre poste du réseau.

```
$ rdate 192.168.3.3
[192.168.3.3]   Mon Apr  8 19:27:27 2002
```

En résumé, une fois que le démon **xinetd** est installé et fonctionnel, l'ajout d'un service se résume à deux actions :

- la mise en place du fichier de configuration sur **/etc/xinetd.d**,
- la copie du programme démon déclaré dans ce fichier sur **/sbin**.

Connexion PPP

On s'est pour l'instant attaché, dans ce chapitre, à la configuration du réseau dans le cas d'une connexion à un réseau local *via* un adaptateur Ethernet. Dans certaines situations pourtant, même isolé de tout réseau local, un système embarqué devra effectuer une connexion temporaire à un réseau distant (comme Internet), et ce en utilisant une simple ligne téléphonique. Le protocole de connexion utilisé est en général PPP (*Point to Point Protocol*). Les distributions Linux récentes fournissent des outils graphiques de configuration d'une connexion PPP, ce qui ne convient pas à une solution embarquée.

Dans cette section, nous allons donc décrire les composants qu'il faut installer et la configuration qui doit être mise en place afin d'ajouter une couche client PPP à notre cible. La mise en place de PPP nécessite quelques conditions au niveau du système.

La validation du support PPP

Celle-ci s'effectue au niveau du noyau Linux, dans le menu *Network device support* des options de configuration du noyau comme indiqué ci-après :

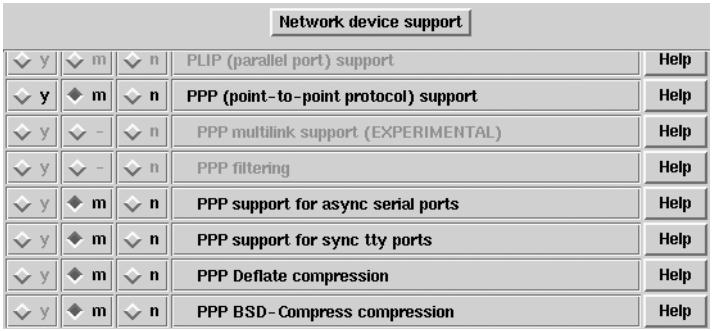


Figure 6-1
Validation du support PPP.

Si vous utilisez une liaison asynchrone, ce qui est le cas le plus fréquent pour un modem connecté sur un port série, il est indispensable de valider l'option *PPP support for async serial port*.

L'installation du programme `pppd`

Ce programme gère le protocole PPP au niveau de l'espace utilisateur de Linux. Nous l'utiliserons ici en mode client (il se connecte sur un serveur PPP distant) mais c'est le même programme qui permettra de construire un serveur PPP avec un système Linux. La version Red Hat 7.2 du programme **`pppd`** utilise quelques bibliothèques supplémentaires que nous devrons installer sur la cible dans **`/lib`**.

```
# ldd /usr/sbin/pppd
        libpam.so.0 => /lib/libpam.so.0 (0x4001c000)
        libdl.so.2 => /lib/libdl.so.2 (0x40024000)
        libutil.so.1 => /lib/libutil.so.1 (0x40028000)
        libcrypt.so.1 => /lib/libcrypt.so.1 (0x4002b000)
        libc.so.6 => /lib/libc.so.6 (0x40059000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Le programme lui-même sera installé sur le répertoire **`/sbin`**.

L'installation du programme `chat`

Ce programme permet de dialoguer avec le modem et d'établir la connexion téléphonique avec le serveur distant. Ce programme utilise une procédure « j'envoie/j'attends » (*expect/send*), ce qui permettra de l'utiliser pour de nombreuses applications de scripts Linux. Le programme sera installé dans le répertoire **`/sbin`**.

La création du point d'entrée `/dev/ppp`

Le point d'entrée doit être créé par la commande :

```
mknod /dev/ppp c 108 0
```

La mise en place du répertoire /etc/ppp

Ce répertoire contient les fichiers de configuration des programmes précédents. Pour les fichiers de configuration, le package PPP installe les fichiers suivants :

```
/etc/ppp/options
/etc/ppp/chap-secrets
/etc/ppp/pap-secrets
```

Le fichier **options** contient les options de **pppd** pour la connexion PPP par défaut. Les fichiers **chap-secrets** et **pap-secrets** contiennent respectivement les identificateurs et mots de passe pour l'utilisation des protocoles CHAP (*Challenge Handshake Authentication Protocol*) et PAP (*Password Authentication Protocol*) auprès du serveur d'accès PPP. Voici un exemple de contenu du fichier **pap-secrets** :

```
# Secrets for authentication using PAP
# client        server  secret                  IP addresses
pficheux        *       mon_mot_de_passe
```

Ce qui indique que, pour tous les serveurs PPP utilisés, le mot de passe PAP de l'utilisateur *pficheux* sera *mon_mot_de_passe*. En réalité, le mot de passe sera crypté à la manière des mots de passes Unix du fichier **/etc/passwd** (voir **man crypt**). Il est également important que ce fichier ne soit lisible que par le super-utilisateur.

```
# ls -l /etc/ppp/pap-secrets
-rw-------    1 root     root          282 fév 11 15:24 /etc/ppp/pap-secrets
```

L'aspect du fichier **chap-secret** est identique.

Le système pourra se connecter à différents serveurs PPP ; la liste des serveurs sera disponible sur **/etc/ppp/peers**, à raison d'un fichier de configuration par serveur.

```
# cat /etc/ppp/peers/free
ttyS0 115200 crtscts usepeerdns noipdefault defaultroute
connect '/sbin/chat -t 60 -v -f /etc/ppp/chat-free'
noauth
lock
idle 120
```

Le fichier indique la ligne utilisée par **pppd** pour se connecter au serveur distant. La configuration courante indique que le client récupère la configuration IP auprès du serveur (**noipdefault** et **usepeerdns**) et met en place le routage par défaut sur la connexion PPP vers le serveur.

On note également qu'il utilise la commande **chat** à laquelle on passe en paramètre le script **chat-free** montré ci-après :

```
ABORT "NO CARRIER"
ABORT "NO DIALTONE"
ABORT "ERROR"
ABORT "NO ANSWER"
ABORT "BUSY"
"" "\d\dat"
OK "at&d2&c1m1"
```

```
OK "atdt0860922000"
CONNECT ""
```

Ce script définit un dialogue de type « j'envoie/j'attends » (*expect/send*) entre le système et le modem. Les lignes ABORT indiquent des conditions d'erreur qui aboutissent à l'interruption de l'appel. En cas de réception finale de la chaîne CONNECT, la connexion modem est considérée comme établie et le programme **pppd** tente d'établir le protocole PPP.

On peut tester la connexion PPP en utilisant la commande **pppd** comme suit :

```
# pppd call free name pficheux
```

En cas d'établissement de la connexion PPP, le fichier **/var/run/ppp0.pid** sera créé et l'on pourra visualiser la nouvelle configuration réseau au moyen des commandes **ifconfig** et **route**.

```
# ifconfig
lo        Lien encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:28 errors:0 dropped:0 overruns:0 frame:0
          TX packets:28 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          RX bytes:1854 (1.8 Kb)  TX bytes:1854 (1.8 Kb)

ppp0      Lien encap:Protocole Point-à-Point
          inet adr:62.147.84.85  P-t-P:192.168.254.254  Masque:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:3
          RX bytes:64 (64.0 b)  TX bytes:97 (97.0 b)
# route
Table de routage IP du noyau
Destination     Passerelle      Genmask         Indic Metric Ref    Use Iface
192.168.254.254 *               255.255.255.255 UH    0      0        0 ppp0
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
default         192.168.254.254 0.0.0.0         UG    0      0        0 ppp0
```

Nous pouvons noter que la règle de routage par défaut est positionnée sur l'interface **ppp0** correspondant à la connexion PPP.

Pour couper la connexion, il suffira de tuer le processus **pppd** par :

```
kill `cat /var/run/ppp0.pid`
```

Il est possible de visualiser la trace de la connexion et de l'établissement du protocole si l'on installe le programme **syslogd** qui est le plus souvent utilisé dans les applications système. Pour cela, il suffit de copier les deux composants, puis de lancer le démon.

```
cp /sbin/syslogd /mnt/emb/sbin
cp /etc/syslog.conf /mnt/emb/etc
/sbin/syslogd
```

Le démon peut bien entendu être lancé systématiquement à partir du script de démarrage **/etc/rc.d/rc.S**. Lorsque le démon est actif, on peut visualiser ce type de trace dans le fichier **/var/log/messages** :

```
Apr 11 15:17:35 duron chat[1355]: atdt0860922000^M^M
Apr 11 15:17:35 duron chat[1355]: CONNECT
Apr 11 15:17:35 duron chat[1355]:  -- got it
Apr 11 15:17:35 duron chat[1355]: send (^M)
Apr 11 15:17:35 duron pppd[1354]: Serial connection established.
Apr 11 15:17:35 duron pppd[1354]: Using interface ppp0
Apr 11 15:17:35 duron pppd[1354]: Connect: ppp0 <--> /dev/ttyS1
Apr 11 15:17:36 duron kernel: PPP BSD Compression module registered
Apr 11 15:17:36 duron kernel: PPP Deflate Compression module registered
Apr 11 15:17:36 duron pppd[1354]: local  IP address 62.147.84.85
Apr 11 15:17:36 duron pppd[1354]: remote IP address 192.168.254.254
Apr 11 15:17:36 duron pppd[1354]: primary   DNS address 213.228.0.168
```

Au moment de l'établissement de la connexion PPP, **pppd** exécute automatiquement le script shell **/etc/ppp/ip-up** qui peut être adapté par l'utilisateur, ce qui permet d'associer l'établissement de la connexion au lancement d'une application. De même, la coupure de la connexion s'accompagnera de l'exécution du script **/etc/ppp/ip-down**.

En résumé

La mise en place d'une connexion réseau est un point très important de la configuration d'un système Linux embarqué. Nous utilisons pour cela les commandes système **ifconfig** et **route**.

L'utilisation de la cible en tant que client de nouveaux services réseau nécessite la mise en place des **libnss** associées, et ce afin de satisfaire à l'architecture de la glibc2.

Une fois que le super-démon **xinetd** est installé, la mise en place de nouveaux services serveur est relativement aisée.

Un système Linux embarqué peut également être connecté à un serveur PPP distant, et ce en ajoutant un minimum de composants au système.

7

Optimisation et mise au point du système

Le chapitre précédent nous a permis de configurer le réseau de notre système Linux embarqué et nous disposons d'ores et déjà d'une version très fonctionnelle. Le présent chapitre va s'attacher à quelques points particuliers comme la configuration du clavier local, la mise en place d'un système d'authentification, la gestion de mémoire flash ou la comparaison des types de systèmes de fichier.

Configuration du clavier

Le clavier de la console Linux est configuré par défaut en QWERTY (clavier anglais/américain). Si vous devez faire des manipulations sur la console, il est intéressant de disposer d'une configuration de clavier AZERTY. Le système de configuration comprend d'une part le programme `loadkeys`, permettant le chargement de la disposition du clavier (carte ou *map*), et d'autre part la carte elle-même, contenue dans un fichier *.kmap* ou *.kmap.gz* (compressé avec `gzip`).

Les fichiers *kmap* se trouvent dans le répertoire `/lib/kbd/keymaps`. Dans les distributions classiques, ce répertoire contient un grand nombre de fichiers correspondant aux différents types de clavier selon les architectures matérielles. Dans notre cas, nous ne copierons que le fichier correspondant à un clavier français/latin1, ainsi que les différents fichiers annexes. Par ailleurs, nous copierons également le programme `loadkeys`, ainsi que le décompresseur `gunzip`.

```
mkdir -p /mnt/emb/lib/kbd/keymaps/i386/azerty
cd /lib/kbd/keymaps
cp i386/azerty/fr-latin1.kmap.gz  i386/azerty
```

```
cp -a -r i386/include i386
cp -a -r include .
cp /bin/loadkeys /mnt/emb/bin
cp /bin/gzip /mnt/emb/bin
cp -a /bin/gunzip /mnt/emb/bin
```

On peut maintenant modifier le fichier **/etc/sysconfig/general** de la cible afin de définir le type de clavier.

```
# General configuration
KEYTABLE=fr
KEYMAP=i386/azerty/fr-latin1
```

Puis on peut ajouter la ligne suivante au fichier **/etc/rc.d/rc.S** de la cible, juste avant le lancement de l'interpréteur de commande :

```
# Clavier
/bin/loadkeys /lib/kbd/keymaps/${KEYMAP}.kmap
```

Au démarrage du système, on obtient alors :

```
Loading /lib/kbd/keymaps/i386/azerty/fr-latin1.kmap.gz
```

Mise en place d'un système d'authentification

Un système Linux est par défaut multi-utilisateur et dispose donc d'un système d'authentification permettant aux utilisateurs référencés de se connecter. Cette authentification est le plus souvent basée sur la saisie d'un nom d'utilisateur ou *login*, suivie de l'entrée d'un mot de passe ou *password*. Dans la version actuelle de notre cible, il n'y a pas d'authentification et le démarrage du système se termine par le lancement direct d'un interpréteur de commande en mode super-utilisateur. Cette configuration n'a qu'un but pédagogique et un système réel aura très souvent besoin d'une fonction d'authentification. Celle-ci ne sera pas forcément utilisée sur une console locale car il faut pour cela que le système puisse disposer d'un écran et d'un clavier. L'authentification sera également utile pour une connexion *via* un terminal série ou bien à travers le réseau *via* une connexion Telnet, SSH ou FTP.

Les premières versions d'Unix utilisaient une authentification à travers une liste locale d'utilisateurs définie dans le fichier **/etc/passwd**. Les versions dérivées de System V R4 (SVR4) utilisent en plus le fichier **/etc/shadow** qui stocke les mots de passe cryptés associés aux utilisateurs définis dans **/etc/passwd**. Les distributions Linux utilisent également une telle configuration. Dans une configuration complexe connectée à un réseau local, l'authentification peut se faire à partir d'une base identique à la base locale, mais centralisée sur un serveur utilisant le protocole NIS (*Network Information Service*) développé par SUN Microsystems.

En ce qui concerne le fonctionnement global de l'authentification, Linux utilise également le principe d'Unix en respectant le schéma suivant :

- Le programme **getty** affiche le message d'invite sur une console. Par défaut, ce message correspond à *login.*
- Lorsque **getty** détecte une entrée de caractère sur la console, il la saisit et la passe au programme **login**.
- Le programme **login** se charge de demander le mot de passe à l'utilisateur en affichant par défaut la chaîne Password.
- En cas de saisie correcte du mot de passe, l'interpréteur de commande associé à l'utilisateur, et défini dans **/etc/passwd**, est exécuté.
- En cas de saisie incorrecte, un message Login incorrect est affiché et le processus recommence.

Le lancement du programme **getty** (appelé **agetty** sous Linux) est déclaré dans le fichier **/etc/inittab**.

```
c1:1235:respawn:/sbin/agetty 38400 tty1 linux
c2:1235:respawn:/sbin/agetty 38400 tty2 linux
```

tty1 et **tty2** correspondent à deux consoles « virtuelles », sur un même écran. On passe d'une console à l'autre en utilisant les combinaisons de touches Ctrl-Atl-F1 et Ctrl-Alt-F2. Pour ajouter une nouvelle console virtuelle, il suffit d'ajouter une ligne au fichier **/etc/inittab** puis de redémarrer le système ou bien de taper **init q** pour forcer le processus **init** à relire son fichier de configuration.

Le système d'authentification décrit ici avait dans sa version initiale une assez forte limitation car la mise en place d'une nouvelle politique d'authentification nécessitait la modification des outils associés comme **login** et **passwd**. Le problème a été résolu de manière élégante en utilisant un système appelé PAM (*Pluggable Authentication Module*) qui permet de modifier le comportement de l'authentification en ajoutant simplement des bibliothèques dynamiques appelées « modules PAM ». Grâce à PAM, il est possible d'ajouter une méthode quelconque d'authentification en ajoutant au système une bibliothèque dynamique et un fichier de configuration. De nombreux modules existent déjà pour s'authentifier auprès d'annuaires de type LDAP ou équivalent.

De ce fait, les programmes comme **login** utilisent, outre celles qui sont habituelles, quelques autres bibliothèques, partagées.

```
# ldd /bin/login
        libcrypt.so.1 => /lib/libcrypt.so.1 (0x4002d000)
        libpam.so.0 => /lib/libpam.so.0 (0x4005a000)
        libdl.so.2 => /lib/libdl.so.2 (0x40062000)
        libpam_misc.so.0 => /lib/libpam_misc.so.0 (0x40066000)
        libc.so.6 => /lib/i686/libc.so.6 (0x40069000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

De même, il est nécessaire d'installer sur le système un certain nombre de fichiers de configuration et bibliothèques partagées situées sur les répertoires **/etc/pam.d** et **/lib/security**.

Il est bien entendu possible de dupliquer ce comportement sur notre cible embarquée mais, dans un éternel souci de réduction de l'empreinte mémoire, nous allons considérer que le système PAM n'est pas utile dans notre cas. Pour ce faire, il sera bien entendu nécessaire de générer de nouvelles versions des programmes concernés ne faisant pas référence au système PAM. Nous devons pour cela installer les archives RPM sources de la distribution Red Hat 7.2, ou bien travailler directement sur les sources des programmes. Une rapide recherche par **rpm** nous indique les noms de paquetages RPM associés au programme **login**.

```
# rpm -qf /bin/login
util-linux-2.11f-9
```

Après récupération du paquetage source associé, nous l'installons sur notre système de développement.

```
rpm -ivh util-linux-2.11f-9.src.rpm
```

Le paquetage en question est maintenant installé dans l'arborescence RPM du système, soit **/usr/src/redhat**. L'archive des sources et les « patches » à appliquer avant compilation sur Red Hat 7.2 sont disponibles sur le sous-répertoire **SOURCES**. L'arborescence des sources, non « patchée » pour l'instant, est disponible sur le sous-répertoire **BUILD**. Le fichier de spécification RPM est disponible sur le sous-répertoire **SPECS**. Pour appliquer les patches, il faut déjà utiliser la commande :

```
rpm -bp /usr/src/redhat/SPECS/util-linux.spec
```

qui va générer une arborescence compatible avec Red Hat 7.2 sur le répertoire **BUILD**. Il faut maintenant modifier les options de compilation du paquetage afin de supprimer le support PAM (HAVE_PAM). On peut également modifier le support des mots de passes masqués (*shadow password*), ce qui permettra de limiter la liste des utilisateurs et mots de passe au fichier **/etc/passwd**.

```
cd /usr/src/redhat/BUILD/util-linux-2.11f
```

En éditant le fichier MCONFIG, on peut modifier les options de compilation du support PAM.

```
# If HAVE_PAM is set to "yes", then login, chfn, chsh, and newgrp
# will use PAM for authentication. Additionally, passwd will not be
# installed as it is not PAM aware.
HAVE_PAM=no

# If HAVE_SHADOW is set to "yes", then login, chfn, chsh, newgrp, passwd,
# and vipw will not be built or installed from the login-utils
# subdirectory.
HAVE_SHADOW=no
```

Le répertoire **login-utils** contient les sources des programmes **login**, **agetty** et **passwd** ; il suffit donc de compiler puis de copier les programmes au bon endroit :

```
cd login-utils
make agetty login passwd
```

```
cp agetty /mnt/smb/sbin
cp login passwd /mnt/emb/bin
```

Au niveau du fichier **/etc/inittab**, et en plus de l'ajout de l'appel à **agetty**, il faut indiquer au système de démarrer en mode multi-utilisateur (*multi-user*) et non plus en mono-utilisateur (*single-user*). Pour cela, on modifie le début du fichier.

```
# Default runlevel.
id:3:initdefault:
# System initialization (runs when system boots).
si:S:sysinit:/etc/rc.d/rc.S

# Script to run when going multi user.
rc:123456:wait:/etc/rc.d/rc.M
```

On ajoute un nouveau script **rc.M** qui pourra contenir des actions à lancer au passage en multi-utilisateur. Pour l'instant, le script contient seulement :

```
#!/bin/sh
# Script to run in multi-user mode
echo "Multi-user mode."
```

Après redémarrage du système, on obtient à la fin :

```
INIT: Entering runlevel: 3
Multi-user mode.
localhost.localdomain login:
```

Remarque

Dans le cas de l'utilisation de BusyBox, le système d'authentification est intégré et peut être mis en place en configurant l'entrée « Login/Password Management Utilities » de la configuration BusyBox accessible par make `menuconfig`.

Configuration des disques flash

Nous avons jusqu'à présent utilisé des disques durs ou disques flash munis d'une interface standard IDE. Ce choix est très avantageux dans la majorité des cas car il ne nécessite pas de configuration particulière du noyau Linux pour le pilotage de ces périphériques. L'interface IDE entraîne cependant un surcoût au niveau du prix du matériel et il est parfois judicieux d'utiliser des mémoires flash ne recourant pas à l'IDE. Dans cette section, nous allons décrire la configuration du noyau pour le pilotage de disque flash M-Systems de type DOC2000 (ou *Disk On Chip*). La principale raison en est simplement la large diffusion de ce produit dans le monde des applications embarquées ; nombreux sont en effet les constructeurs de cartes mère industrielles qui fournissent un emplacement destiné à ce type de composant. La capacité du composant va de 16 à 568 Mo. Nous décrirons aussi brièvement la configuration du pilote MTD pour les mémoires flash de type CFI (*Common Flash Interface*).

La configuration du noyau pour la DOC2000 peut s'effectuer de deux manières :

- en utilisant le pilote fourni par M-Systems sous forme d'un « patch » du noyau,
- en utilisant le driver MTD du noyau 2.4.

Attention

Le pilote MTD fonctionne avec les flash DOC2000 et Millenium mais ne fonctionne *pas* avec les flash Millenium Plus. Pour ces dernières, il est impératif d'utiliser le pilote fourni par M-Systems.

Utilisation du pilote M-Systems

Le pilote est disponible en téléchargement sur le site de M-Systems à l'adresse *http://www.m-sys.com*. Les composants sont répartis sur deux paquetages :

- un paquetage contenant des utilitaires MS-DOS destinés au formatage bas niveau de la flash ainsi que le *BIOS driver* M-Systems (`tffs_dostools_5041.zip` qui contient en particulier `DFORMAT` et `DINFO`) ;
- un paquetage contenant le patch du noyau (pour les versions 2.0, 2.2 et 2.4), la documentation d'installation ainsi que quelques utilitaires comme une version adaptée de LILO (`doc-linux-5.0.0.tgz`).

Les utilitaires DOS nécessitent l'installation d'un système d'exploitation compatible. Si l'on ne dispose pas de licence MS-DOS, on peut utiliser le clone DR-DOS/OPEN-DOS disponible en téléchargement auprès de la société Caldera sur l'URL *http://www.drdos.net*. Cette adresse permet de télécharger les images des trois disquettes d'installation. Les disques flash de type DOC 2000 sont fournis déjà formatés avec une partition de type DOS déjà créée. Pour formater le disque flash, il suffit d'utiliser la commande `DFORMAT` fournie dans l'archive des utilitaires DOS M-Systems.

```
DFORMAT /WIN:D000 /S:DOC504.EXB
```

Le fichier EXB correspond au *firmware* installé sur le disque flash. Une description complète des utilitaires DOS est disponible dans le document fourni avec l'archive `tffs_dostools_5041.zip`.

L'extraction de la deuxième archive crée le répertoire `doc-linux-5.0.0` qui contient le fichier `README.install` décrivant en détail l'installation du pilote sous Linux. Pour appliquer le patch aux sources du noyau, il suffit de faire :

```
cd doc-linux-5.0.0/drivers
./patch_linux linux-2_4-patch driver-patch
./mknod_fl
```

Si le répertoire `/usr/src/linux` n'existe pas (ce qui est souvent le cas sur un système équipé d'un noyau 2.4), il faudra passer le répertoire des sources du noyau à modifier en troisième paramètre du script `patch_linux`. La dernière ligne permet de créer les points d'entrées correspondant au pilote de la flash dans le répertoire `/dev`, soit :

```
# ls -l /dev/msys
total 0
brw-------    1 root     root     100,   0 avr 14 21:47 fla
brw-------    1 root     root     100,   1 avr 14 21:47 fla1
brw-------    1 root     root     100,   2 avr 14 21:47 fla2
brw-------    1 root     root     100,   3 avr 14 21:47 fla3
brw-------    1 root     root     100,   4 avr 14 21:47 fla4
brw-------    1 root     root     100,  64 avr 14 21:47 flb
...
```

Lorsque le patch est appliqué, il suffit de valider le support *M-Systems DOC device support* dans le menu *Block devices* de la configuration du noyau Linux. Si le disque flash est utilisé comme périphérique de démarrage, vous pouvez dans un premier temps valider le support dans la partie statique du noyau (cocher *y* et non *m*). Sachez cependant que la diffusion d'un tel noyau n'est pas conforme à la licence du noyau Linux (la GPL) car le noyau statique généré est constitué de code GPL auquel est ajouté du code non-GPL (venant de M-Systems). Pour être conforme à la GPL, vous devrez utiliser le support par modules, ce qui oblige à créer un disque mémoire de démarrage ou *initrd*. L'utilisation de la fonction **`initrd`** sera décrite au chapitre suivant.

Lorsque le noyau est recompilé puis installé comme décrit au chapitre 4, on peut d'ores et déjà redémarrer le système et vérifier la détection du disque flash dans les messages de démarrage, ou après coup en utilisant la commande **`dmesg`**.

```
Flash disk driver for DiskOnChip2000
Copyright (C) 1998,2001 M-Systems Flash Disk Pioneers Ltd.
DOC device(s) found: 2
Fat Filter Enabled
fl_geninit: registered device at major: 100
```

Le disque est dès maintenant accessible comme un disque classique en utilisant le device /**`dev/msys/fla`** (pour le premier disque flash détecté). Le disque contient par défaut une partition MS-DOS qu'il faudra certainement remplacer par une partition Linux en recourant à l'utilitaire **`fdisk`** décrit au chapitre 5.

```
# fdisk /dev/msys/fla
Command (m for help): p
Disk /dev/msys/fla: 16 heads, 4 sectors, 1001 cylinders
Units = cylindres of 64 * 512 bytes
        Device Boot    Start       End    Blocks   Id  System
/dev/msys/fla1   *         1      1001     32030   83  Linux

Command (m for help):
```

À partir de là, on peut créer n'importe quel type de partition (ext2, ext3, etc.) sur le device **`/dev/msys/fla1`**. La partition peut ensuite être montée comme suit :

```
mount -t ext3 /dev/msys/fla1 /mnt/flash
```

On peut également démarrer à partir du disque flash en utilisant la version spéciale de LILO fournie sur l'archive M-Systems, soit sous forme de paquetages RPM, soit sous

forme d'archives au format **tar.gz**. Cette version spéciale est nécessaire car elle prend en compte la géométrie spéciale du disque flash M-Systems, ce qui n'est pas le cas dans la version standard de LILO.

```
# cd doc-linux-5.0.0/lilo/
# ls -l
total 484
-rw-rw-rw-    1 root     root      668 aoû  1  2001 README.lilo
-rw-r--r--    1 root     root    34236 jui 30  2001 doc-lilo-0.21-19.i386.rhat52.rpm
-rw-r--r--    1 root     root    35369 jui 30  2001 doc-lilo-0.21-19.i386.rhat62.rpm
-rw-r--r--    1 root     root   190516 jui 30  2001 doc-lilo-0.21-19.src.rpm
-rw-r--r--    1 root     root    32223 jui 30  2001 lilo-bin.21.tar.gz
-rw-r--r--    1 root     root   183621 jui 30  2001 lilo-src.21.tar.gz
```

Avec l'installation d'un paquetage RPM binaire, on dispose du programme **/sbin/doc-lilo** et du secteur de boot spécial **doc.b** qui remplace le fichier standard **boot.b** utilisé par LILO, comme décrit au chapitre 4.

```
# rpm -ql doc-lilo
/boot/doc.b
/sbin/doc-lilo
```

Un fichier **lilo.conf** adapté au disque flash sera donc du style :

```
boot=/dev/msys/fla
install=/boot/doc.b
map=/boot/map
read-only
vga = normal
# End LILO global section
image = /boot/bzImage-2.4.18_msys
        root = /dev/msys/fla1
        label = linux
```

Dans certains cas, il sera parfois nécessaire de forcer le numéro du disque (paramètre **bios**) par les lignes suivantes dans les paramètres globaux :

```
disk=/dev/msys/fla
bios=0x80
```

Si la partition du disque est montée sur **/mnt/flash** et donc le fichier **lilo.conf** présent sur **/mnt/flash/etc**, on installera LILO sur le secteur de démarrage du disque flash en utilisant la commande :

```
/sbin/doc-lilo -r /mnt/flash -v
```

Après déconnexion des autres périphériques de boot plus prioritaires et redémarrage du système, on devrait démarrer sur le disque flash. Il est bien entendu nécessaire de modifier préalablement les fichiers de configuration qui dépendent du nom du périphérique contenant la partition principale (*root filesystem*), en particulier le fichier **/etc/fstab** présent sur le disque flash.

Attention

Il est également indispensable de créer les entrées spéciales /dev/msys sur le répertoire /mnt/flash/dev *avant* de démarrer le système sur le disque flash.

Certaines versions de LILO sont incompatibles avec le BIOS M Systems installé par DFORMAT. Il est alors conseillé de mettre à jour LILO avec une version plus récente. Les sources de LILO sont disponibles à l'adresse *ftp://sunsite.unc.edu/pub/Linux/system/boot/lilo*. La description du problème et des solutions se trouve dans le fichier README.install contenu dans l'archive doc-lilo-5.0.0.tgz.

Utilisation du pilote MTD

Le projet MTD (*Memory Technology Device*) est mené par David Woodhouse (*dwm2 @infradead.org*) et la page d'accueil est disponible sur *http://www.linux-mtd.infradead.org*. Ce pilote a l'avantage d'être totalement Open Source et de supporter un grand nombre de disques flash, dont les produits M-Systems, à l'exception du modèle Millenium Plus. Le but du projet est de mettre en place une interface simple entre le pilote matériel de la flash et les couches supérieures du noyau Linux. Ce projet est également couplé au développement du système de fichier JFFS2 (*Journalling Flash File System* version 2). MTD est intégré à l'arborescence des noyaux 2.4 et 2.6.

Le seul défaut du projet MTD est un manque relatif de documentation concernant la configuration et l'utilisation des composants. Un fichier, **`mtd-jffs-HOWTO.txt`**, reprend cependant les procédures d'installation et de configuration. Il est disponible depuis la page d'accueil du projet.

Les dernières versions contenant les pilotes et divers utilitaires sont disponibles depuis le serveur *CVS* du projet ou bien sous forme d'une copie journalière au format **`tar.gz`** (*daily snapshot*) depuis la page d'accueil. Le projet est en perpétuelle évolution et il est donc conseillé de télécharger la dernière version auprès du serveur CVS plutôt que d'utiliser celle fournie dans le noyau Linux 2.4 ou 2.6. Il faut cependant noter que seule la version 2.6 est maintenue à ce jour. Pour ce faire, il faut copier l'arbre CVS en local au moyen des commandes :

```
cd /usr/src
cvs -d :pserver:anoncvs@cvs.infradead.org:/home/cvs login
```

Après saisie du mot de passe *anoncvs*, il faut faire :

```
cvs -d :pserver:anoncvs@cvs.infradead.org:/home/cvs co mtd
```

ce qui a pour effet de créer un répertoire **`/usr/src/mtd`**. On doit ensuite se positionner dans le répertoire **`patches`** afin de modifier les sources du noyau courant.

```
cd /usr/src/mtd/patches
sh patchin.sh /usr/src/linux-2.4
```

La configuration du noyau pour l'utilisation de MTD est décrite dans le fichier **`mtd-jffs-HOWTO.txt`**. Si nous devons utiliser le disque flash comme périphérique de démarrage, il sera plus simple de valider le support MTD dans la partie statique du noyau. La figure ci-après indique la configuration à effectuer dans le menu principal du pilote MTD.

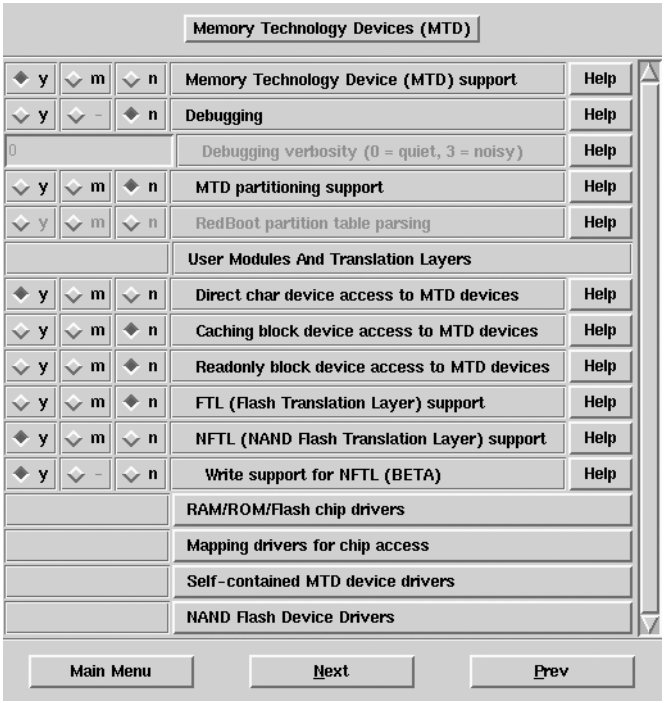


Figure 7-1
Configuration principale de MTD.

En cliquant sur l'option *Self-contained MTD device drivers*, ce qui correspond entre autres aux disques flash M-Systems ; on obtient l'écran de configuration suivant :

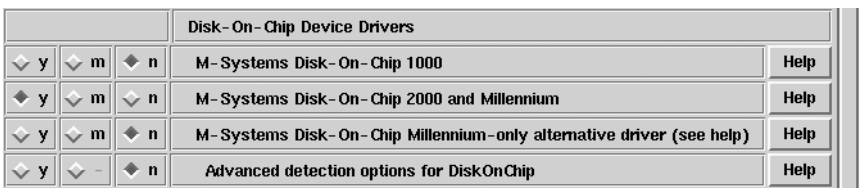


Figure 7-2
Validation du support DOC 2000.

La validation du support du système de fichiers JFFS2 est quant à elle effectuée dans le menu *File systems* du menu principal de configuration du noyau, comme indiqué sur la figure ci-après :

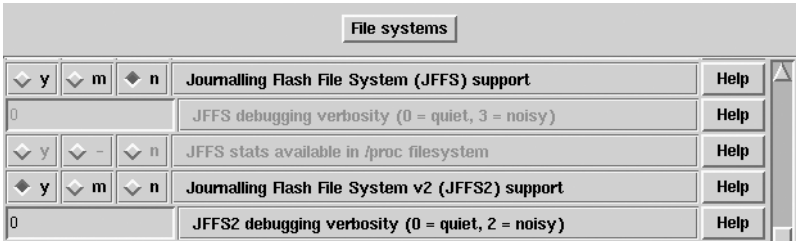


Figure 7-3
Validation du support JFFS2.

Comme pour le pilote M-Systems, il est nécessaire de créer les points d'entrée correspondants dans le répertoire **`/dev`**. On utilise pour cela le script **`MAKEDEV`** fourni dans le répertoire **`util`** des sources du projet MTD.

Après compilation et installation du nouveau noyau, le redémarrage du système doit provoquer l'affichage d'un message, tel que celui-ci :

```
DiskOnChip 2000 found at address 0xD0000
Flash chip found: Manufacturer ID: 98, Chip ID: 73 (Toshiba TH58V128DC)
2 flash chips found. Total DiskOnChip size: 32 MiB
```

Le pilote MTD permet également d'obtenir des informations *via* le système de fichiers virtuel **`/proc`**.

```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 02000000 00004000 "DiskOnChip 2000"
```

Comme pour le pilote M-Systems, on peut manipuler le disque flash avec les utilitaires classiques.

```
# fdisk /dev/nftla
Command (m for help): p
Disk /dev/nftla: 16 heads, 4 sectors, 1001 cylinders
Units = cylindres of 64 * 512 bytes

     Device Boot    Start       End    Blocks   Id  System
/dev/nftla1   *         1      1001     32030   83  Linux

Command (m for help):
```

Une fois la partition créée, on peut alors créer un système de fichier ext2 ou ext3 en utilisant la commande **`mke2fs`** sur le device **`/dev/nftla1`**, qui est un device de type bloc. La création d'un système JFFS2 utilise le device de type caractère **`/dev/mtd0`** et sera décrite dans la prochaine section concernant les systèmes de fichiers. Le montage de la partition ne présente pas de difficulté.

```
# mount -t ext3 /dev/nftla1 /mnt/flash
```

Le boot sur le disque flash nécessite là encore une version modifiée de LILO. Une archive du LILO modifié est disponible sur le répertoire **patches** des sources du projet MTD.

```
# tar tzvf patches/lilo-mtd.tar.gz
-rw-r--r-- dvir/dvir      4540 2000-03-18 16:43:00 boot.b-mtd
-rwxr-xr-x dvir/dvir     51500 2000-03-19 17:44:00 lilo-mtd
-rw-r--r-- dvir/dvir      2361 2000-03-04 19:51:00 lilo21-mtd-patch
```

Un fichier **lilo.conf** adapté est très similaire au fichier précédent.

```
boot=/dev/nftla
install=/boot/boot.b-mtd
map=/boot/map
read-only
vga = normal
# End LILO global section
image = /boot/bzImage-2.4.18_mtd
        root = /dev/nftla1
        label = linux
```

On installe le secteur de démarrage avec la commande :

```
# /sbin/lilo-mtd -r /mnt/flash -v
Patched LILO for M-Systems' DiskOnChip 2000
LILO version 21, Copyright 1992-1998 Werner Almesberger

Reading boot sector from /dev/nftla
Warning: /dev/nftla is not on the first disk
Merging with /boot/boot.b-mtd
Boot image: /boot/bzImage-2.4.18_mtd
Added linux *
Backup copy of boot sector in /boot/boot.5D00
Writing boot sector.
```

Attention

Il est également indispensable de dupliquer sur /mnt/flash/dev les nœuds créés par le script MAKEDEV *avant* de démarrer le système sur le disque flash.

Les mémoires flash CFI (Common Flash Interface)

Le pilote MTD précédemment décrit permet également de piloter les mémoires de type CFI. Le standard CFI a été développé par Intel et permet d'utiliser des mémoires de marques différentes avec la même couche logicielle. Une introduction au standard CFI est disponible en ligne chez Intel à l'adresse *http://developer.intel.com/design/flcomp/cfi_bg.htm*.

L'utilisation nécessite le support générique MTD comme dans le cas des DOC 2000. En plus de cela, il faut sélectionner le type de flash en cliquant sur RAM/ROM/Flash chip driver dans la page principale de configuration MTD. On obtient l'écran suivant :

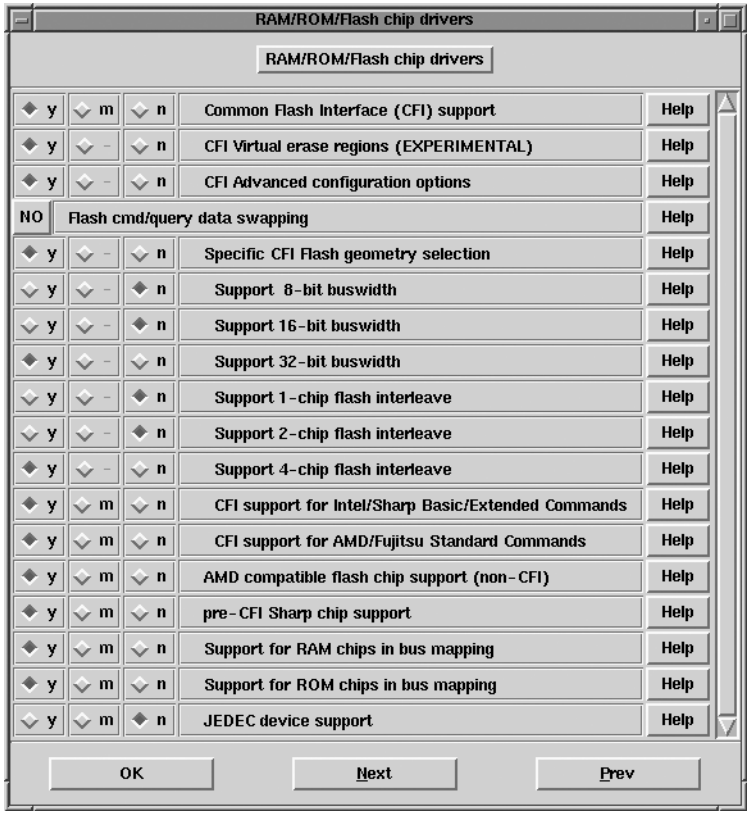


Figure 7-4
Sélection du type de flash.

sur lequel on devra valider le type de flash et éventuellement la géométrie. Si la flash est visible dans l'espace mémoire du processeur, on devra sélectionner les caractéristiques de l'adressage en cliquant sur *Mapping drivers for chip access* dans la page principale de configuration MTD.

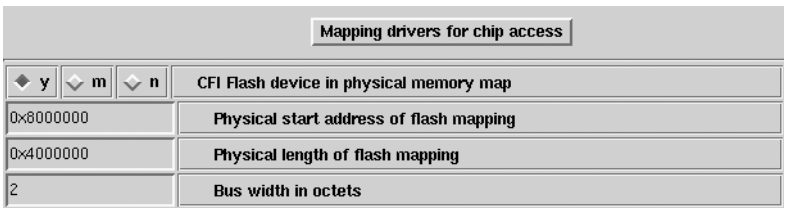


Figure 7-5
Adressage de la flash.

Utilisation d'une clé USB

La clé USB est un périphérique de plus en plus populaire et dont le coût n'a cessé de baisser ces dernières années. On peut aujourd'hui trouver des clés USB de bonne capacité pour quelques dizaines d'euros au point de constituer un support marketing de choix. Certaines sociétés commercialisent des clés USB déjà équipées du système Linux (voir *http://www.flonix.com*) mais il est relativement aisé de mettre en place une distribution réduite sur un tel périphérique. Une telle distribution peut être très intéressante à des fins de démonstration, de tests ou d'outils de maintenance car on pourra alors avoir sa propre distribution dans la poche !

Une clé USB est vue sous Linux comme un disque SCSI. La plupart des distributions actuelles intègrent le support des clés USB en natif (dans le noyau livré) et l'insertion de la clé provoque l'affichage du message suivant dans les traces du système.

```
Sep 21 12:13:26 localhost kernel: hub.c: new USB device 00:07.2-1, assigned address 2
Sep 21 12:13:26 localhost kernel: usb.c: USB device 2 (vend/prod 0xea0/0x6803)
➥is not claimed by any active driver.
Sep 21 12:13:27 localhost kernel: SCSI subsystem driver Revision: 1.00
Sep 21 12:13:27 localhost kernel: Initializing USB Mass Storage driver...
Sep 21 12:13:27 localhost kernel: usb.c: registered new driver usb-storage
Sep 21 12:13:27 localhost kernel: scsi0 : SCSI emulation for USB Mass Storage devices
Sep 21 12:13:27 localhost kernel: Vendor: OTi       Model: Flash Disk        Rev: 1.11
Sep 21 12:13:27 localhost kernel: Type:   Direct-Access         ANSI SCSI revision: 02
Sep 21 12:13:27 localhost kernel: USB Mass Storage support registered.
```

La clé est utilisable à travers le fichier spécial **`/dev/sda`** et l'on peut bien entendu la partitionner, formater, monter, etc. La clé suivante contient deux partitions de 16 Mo, l'une au format VFAT pour les transferts Windows, l'autre au format EXT3 contenant une mini-distribution Linux. L'existence de cette deuxième partition Linux ne trouble en rien l'utilisation de la clé sous Windows.

Le démarrage sur la clé pose quelques petits problèmes :

- Le BIOS du PC doit permettre le démarrage sur support disque USB. C'est le cas de la majorité des PC spécialisés pour les applications embarquées (EDEN, LEX, etc.) mais ce n'est pas le cas des PC grand public.
- La détection de la clé USB lors du démarrage du noyau Linux est asynchrone par rapport au montage du système de fichiers principal (ou *root filesystem*). De ce fait, un noyau Linux standard ne pourra pas utiliser comme *root filesystem* une partition d'une clé USB. Cependant la modification à apporter au noyau pour permettre cette utilisation est minime et se limite à un patch de quelques lignes appliqué au fichier **`init/do_mounts.c`**. Il existe plusieurs patches disponibles, celui-ci est accessible depuis le FAQ du site *http://www.linux-usb.org*. Ce problème est également présent pour le noyau 2.6 et le même type de patch sera applicable.

Le principe de la modification est simple : on ajoute une attente de quelques secondes (maximum 5 essais) pour laisser le temps à la clé USB d'être détectée. Ce n'est pas très élégant mais c'est simple et cela fonctionne. Les quelques lignes de code à ajouter à la fin

de la fonction **mount_root()** sont présentées ci-dessous. Il existe d'autres patches disponibles et accessibles depuis la FAQ du site *http://www.Linux-usb.org*.

```
/*
 * Patch for USB boot
 */
          {
            /* begin jordi ************************** */
            static DECLARE_WAIT_QUEUE_HEAD (jordi_queue);
            printk ("\n\n\n--------------------------------\n");
            printk (" WAITING FOR A WHILE (1000) \n");
            printk (" TO DETECT THE USB DISK \n");
            sleep_on_timeout (&jordi_queue, 1000);
            printk ("--------------------------------\n\n\n");
            /* end jordi *************************** */
          }
/* End of patch */
        mount_block_root("/dev/root", root_mountflags);
}
```

Au niveau de la configuration du noyau, il faudra valider les différentes options (SCSI et USB) en statique pour permettre la détection de la clé USB en tant que disque de démarrage. Les options à valider sont simples :

- Valider *SCSI support* et *SCSI disk support* dans le menu *SCSI support.*
- Valider *Support for USB*, le type de contrôleur USB (*OHCI* ou *UHCI*) ainsi que *USB mass storage support* dans le menu *USB support.*

Au niveau du fichier **/etc/fstab**, on fera référence au fichier spécial **/dev/sda1** pour monter le système de fichiers principal.

```
/dev/sda1        /      ext3    defaults        1 1
```

Concernant l'installation de LILO, le principe est le même que pour les disques durs IDE (puisque c'est également un périphérique en mode bloc) sauf qu'il faut utiliser **/dev/sda** au lieu de **/dev/hda** :

```
boot=/dev/sda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
default=linux

image=/boot/bzImage-2.4.27-usb
        label=linux
        read-only
        root=/dev/sda1
```

Les différents types de systèmes de fichiers

Nous avons installé sur notre cible le système de fichier ext3, dérivé de ext2 et très fréquemment utilisé sur Linux. Pour un système embarqué, nous rappelons que la contrainte d'un système de fichier « journalisé », donc résistant aux arrêts imprévus, est très importante.

Ext2/ext3

Le système de fichier ext3 est l'extension journalisée du système de fichier ext2 qui est le type le plus répandu sur les systèmes Linux. Historiquement, ext2 a été développé au MASI, ex-laboratoire de l'université Jussieu de Paris, par Rémy Card, pour une version universitaire d'un système Unix *like* appelé *MASIX*. Les deux systèmes sont compatibles et l'on peut même utiliser une partition ext3 sur un noyau supportant uniquement ext2, mis à part le journal, qui ne sera pas pris en compte. Nous avons utilisé ext3 au chapitre 5 car il est d'un emploi très aisé sur n'importe quel périphérique en mode bloc. Il est intégré à l'arborescence des sources du noyau Linux 2.4 et maintenu par la société Red Hat.

ReiserFS

Contrairement à ext3, ReiserFS fut conçu dès le départ comme un système de fichier journalisé. De conception plus récente, il est très efficace dans le cas de partitions de grandes tailles gérant nombre de petits fichiers. Il a été initialement développé par Hans Reiser à l'université de Stanford à Palo Alto. Hans Reiser a depuis créé une société commerciale autour de ReiserFS (NAMESYS). Bien qu'intégré dans l'arborescence des sources de noyaux Linux 2.4 et 2.6, les dernières versions sont disponibles sur *http://www.namesys.com*. L'utilisation sur la Red Hat 7.2 nécessite l'installation du paquetage `reiserfs-utils`. On peut créer un tel système de fichier sur un périphérique de type bloc en utilisant la commande `mkreiserfs`.

Il est cependant peu adapté à un environnement embarqué car limité à une taille minimale de partition de 32 Mo.

JFFS2

JFFS2 (*Journaling Flash File System, version 2*) est intimement lié au projet MTD décrit dans ce chapitre. Il a pour origine le système JFFS initialement développé par la société suédoise AXIS (*http://developer.axis.com/software/jffs)* sur le noyau 2.0. JFFS et JFFS2 sont maintenant intégrés dans l'arborescence des noyaux 2.4 et 2.6 et les dernières versions sont disponibles sur le serveur CVS du projet MTD décrit précédemment. Outre des améliorations structurelles concernant la gestion des blocs erronés (*bad blocks*) et la récupération d'espace (*garbage collector*), une caractéristique intéressante de JFFS2 par rapport à JFFS est la fonction de compression des données en temps réel. La validation du support de JFFS et/ou JFFS2 se fait au niveau du menu *File systems* de la configuration du noyau.

Pour créer un système de fichier JFFS2, il faut utiliser le programme **mkfs.jffs2** disponible sur la distribution MTD ou sur le serveur CVS dans le répertoire **util**. La procédure est décrite dans le document **mtd-jffs-HOWTO.txt**. En résumé, outre la détection de la mémoire flash décrite aux sections précédentes, la création du système de fichier se résume aux actions suivantes :

```
# mkfs.jffs2 -d /home/jffs2_stuff -o /tmp/jffs2.img
# cp  /tmp/jffs2.img /dev/mtd0
```

La première ligne correspond à la création d'une image du système de fichier dans un fichier unique à partir d'un répertoire contenant l'arborescence à copier. La deuxième ligne copie simplement ce fichier sur le device correspondant à la mémoire flash utilisée. Si la mémoire flash n'a jamais été initialisée, notez qu'il est nécessaire de le faire avec la commande **erase** disponible sur le même répertoire d'utilitaires.

```
# erase /dev/mtd0
```

Pour monter le système de fichier, il suffit de faire :

```
# mount -t jffs2 /dev/mtdblock0 /mnt/flash
```

Notez que l'on utilise le device **mtdblock0** car la commande **mount** nécessite un périphérique en mode bloc.

CRAMFS

CRAMFS (*Compressed ROM File System*) est un système de fichier en lecture seule utilisant l'algorithme de compression de la **zlib** (identique à **gzip**). La taille maximale de chaque fichier est limitée à 16 Mo et la taille totale du système de fichier est limitée à 256 Mo. Pour utiliser CRAMFS, il faut déjà valider le support dans la configuration du noyau dans la rubrique *Filesystems*. CRAMFS utilise une image générée avec l'utilitaire **mkcramfs** qui permet de créer un fichier image à partir d'un répertoire. Les sources de l'utilitaire sont disponibles à l'adresse *http://cvs.bofh.asn.au/cramfs*.

```
# mkcramfs MicroW microw.img
Directory data: 14092 bytes
Everything: 4252 kilobytes
Super block: 76 bytes
CRC: 8647fdf8
```

Le fichier image pourra ensuite être monté en utilisant la fonction de *loopback* du noyau Linux. Il faut pour cela avoir validé en module ou en statique l'option dans le menu *Block devices/Loopback device support* de la configuration du noyau. Cette fonction permet de monter un fichier image (par exemple, un fichier image ISO d'un CD-Rom) exactement comme si l'on montait le support réel. Dans le cas du fichier CRAMFS, on fera :

```
mount -t cramfs -o loop microw.img /mnt/cramfs
```

si **/mnt/cramfs** est un point de montage valide.

Utilisation des disques mémoire

Le noyau Linux offre la possibilité de travailler sur des disques mémoire ou *ramdisks*. L'utilisation d'un disque mémoire a de nombreux avantages, dont la rapidité d'accès aux données mais aussi le faible coût de la RAM par comparaison avec d'autres supports physiques comme la mémoire flash. Pour utiliser des disques mémoire, il faut déjà valider le support dans le noyau Linux avec le menu *Block devices* comme décrit dans la figure ci-après :

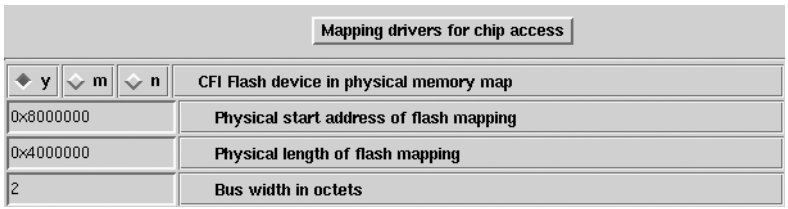


Figure 7-6

Support des disques mémoire.

Le principe du noyau Linux est d'allouer automatiquement un certain nombre de disques de taille fixe, cette taille étant par défaut définie dans le noyau. La valeur par défaut est de 4096 octets. On peut modifier cette valeur dans la configuration présentée ci-après, mais également passer la nouvelle valeur lors du démarrage du système à travers le chargeur LILO :

```
LILO: linux ramdisk_size=8192
```

Comme décrit précédemment, cette configuration pourra être ajoutée au fichier **`/etc/lilo.conf`** grâce à une directive **`append`** :

```
append="ramdisk_size=8182"
```

Chaque disque mémoire est vu à travers un pilote de type bloc **`/dev/ramX`**, X variant de 0 à 20. Ce device est utilisable comme n'importe quel périphérique de stockage en mode bloc. On pourra par exemple créer un système de fichier d'une taille de 2 Mo en faisant :

```
# dd if=/dev/zero of=/dev/ram2 bs=1k count=2048
# mke2fs -vm0 /dev/ram2 2048
```

Le système peut ensuite être monté et utilisé comme n'importe quel système de fichier.

```
# mount -t ext2 /dev/ram2 /mnt/tmp
# df /mnt/tmp
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/ram2                 2011        13      1998   1% /mnt/tmp
```

On peut donc imaginer de remplir ce système de fichier avec les composants d'un *root file system* minimal comme nous l'avons fait au chapitre 5. Une fois le système créé, il est possible de compresser le résultat en un fichier unique. Dans l'exemple qui suit, nous

considérons que le système de fichier en question est suffisamment petit pour tenir au maximum sur une disquette après compression avec **gzip**.

```
dd if=/dev/ram2 bs=1k count=2048 | gzip -v9 > /tmp/ram_image.gz
```

Nous pouvons ensuite créer une disquette de démarrage (*bootable*) à partir d'un noyau sur lequel nous aurons validé le support des disques mémoire.

```
dd if=bzImage of=/dev/fd0 bs=1k
```

Il faut ensuite copier l'image compressée du système de fichier sur la même disquette (et sur une autre, si l'espace restant n'est pas suffisant). Si l'on estime que le noyau occupe au maximum 400 Ko et que l'espace doit être suffisant sur la même disquette, on peut copier l'image après le noyau en faisant :

```
dd if=/tmp/ram_image.gz of=/dev/fd0 bs=1k seek=400
```

On indique ensuite au noyau que le root file system sera lu sur une disquette, et ce au moyen de la commande **rdev**.

```
rdev /dev/fd0 /dev/fd0
```

Il reste à indiquer au noyau la localisation du système de fichier à l'aide de la même commande. La manipulation du disque mémoire avec **rdev** est effectuée grâce à l'option **-r** à laquelle on passe le nom du device contenant l'image du disque mémoire (ici, **/dev/fd0** pour la disquette) suivie de la valeur d'un masque décimal de paramétrage. Le tableau suivant donne la signification des valeurs.

Tableau 7-1. Paramètres de rdev -r

Bits	Paramètre	Description
0 à 10	ramdisk_start	Position dans le support
11 à 13	Non utilisé	Fixé à 0
14	load_ramdisk	Chargement du ramdisk
15	prompt_ramdisk	Arrêt avant chargement

Les noms des paramètres indiqués correspondent à ceux que LILO devrait passer au noyau pour obtenir le même résultat que la commande **rdev**.

Les bits 0 à 10 indiquent la position de l'image compressée du disque mémoire par rapport au début du support. Dans notre cas, la valeur est 400. Si l'image est sur une autre disquette, la valeur est 0.

Le bit 14 indique au noyau qu'un disque mémoire doit être chargé et le bit 15 indique au noyau de demander confirmation à l'utilisateur avant de charger le disque mémoire afin de permettre le changement de support si nécessaire. Si nous validons ces 2 bits nous obtenons 2^15 + 2^14 + 400 = 49 552. Si l'arrêt n'est pas nécessaire la valeur sera 2^14 + 400 = 16 784. On en déduit la commande.

```
rdev -r /dev/fd0 16784
```

Dans le cas d'une image copiée sur une deuxième disquette la valeur sera 2^15 + 2^14 + 0 = 49 152, soit la commande :

```
rdev -r /dev/fd0 49152
```

ce qui correspond à une ligne de commande LILO du type :

```
ramdisk_start=0 load_ramdisk=1 prompt_ramdisk=1
```

La technique du disque mémoire est fréquemment utilisée pour la fonction **initrd** qui permet à un noyau de monter un système de fichier principal (*root file system*) initial à partir d'un disque mémoire. Le but est de définir un démarrage en deux phases, une première permettant d'accéder à tous les périphériques (par exemple, un disque SCSI de démarrage) et une seconde phase utilisant un noyau dont la partie statique est réduite au minimum, donc ne contenant pas les pilotes SCSI. Le nom du fichier correspondant à l'image mémoire à charger est défini dans le fichier **lilo.conf**.

```
image=/boot/vmlinuz-2.4.7-10
        label=linux
        initrd=/boot/initrd-2.4.7-10.img
        read-only
        root=/dev/hda1
```

Le fichier **.img** est compressé avec **gzip**, et l'option **-9**. On peut visualiser son contenu en utilisant la fonction *loopback* décrite précédemment :

```
# file initrd-2.4.7-10.img
initrd-2.4.7-10.img: gzip compressed data, deflated, last modified: Fri Apr 26
➥ 10:46:07 2002, max compression, os: Unix
# gunzip -c initrd-2.4.7-10.img > /tmp/i
# ls -l /tmp/i
-rw-r--r--    1 root     root      3072000 avr 29 17:27 /tmp/i
# mount -t ext2 -o loop /tmp/i /mnt/tmp
# ls /mnt/tmp/
bin  dev  etc  lib  linuxrc  loopfs  proc  sbin  sysroot
```

La racine du système de fichier contient un script **linuxrc** qui est exécuté après le chargement du disque mémoire initial et qui permet normalement de basculer sur le système définitif. Le disque mémoire est ensuite monté sur un point de montage **/initrd**. Quand le script **linuxrc** n'existe pas, le système ne sort jamais du disque mémoire et l'on peut donc envisager d'utiliser cette fonction pour un système embarqué minimal.

Un exemple d'utilisation de CRAMFS et disque mémoire

Si l'on observe la structure d'un système Linux, on s'aperçoit qu'une grande majorité du système de fichiers peut être configurée en lecture seule. Mises à part quelques exceptions que nous décrirons plus loin, les parties nécessitant l'accès en lecture-écriture sont les suivantes :

- le répertoire **/var** contenant des données de fonctionnement souvent volatiles ;

- le répertoire **/tmp** encore plus volatile !
- les répertoires de configuration (exemple : définition d'adresse IP, etc.) ;
- les répertoires d'utilisateurs ou applicatifs (exemple : stockage de données enregistrées par le système). Ces derniers pourront être placés sur un véritable disque dur et il est également possible de monter la partition à la demande afin de limiter les risques.

Outre la dernière catégorie que nous ne traiterons pas ici, il est donc relativement simple de mettre en place l'architecture suivante :

- La majorité du système est sur une partition en lecture seule de type CRAMFS.
- Le répertoire **/var** (point de montage) est sur un disque mémoire.
- Le répertoire **/tmp** est un lien symbolique sur /var/tmp.
- Le répertoire de configuration utilisera un format classique (EXT2, EXT3, JFFS2 ou même Minix). La partition est de très faible taille. On peut même imaginer de stocker cette configuration sur une partition non formatée (au format TAR directement écrit sur le fichier spécial de la partition) ce qui limite les risques de problèmes de système de fichiers dus à la coupure d'alimentation. Dans un système réel on pourra même sauver les fichiers de configuration sur une partition créée sur une mémoire sauvegardée de type SRAM.

À titre d'exemple concret nous supposerons que nous utilisons un DiskOnChip de 8 Mo. Cette configuration correspond au cas réel d'un routeur Wi-Fi construit sur la base d'un PC de type PC Light basé sur un processeur VIA C3. Le principe est d'utiliser trois partitions sur la mémoire flash :

- Une première partition **/dev/nftla1** correspondant à **/boot** et contenant le noyau et les composants de LILO. Il n'est pas nécessaire que cette partition soit montée lors du fonctionnement du système mais seulement lors de la mise à jour éventuelle du noyau depuis un environnement de développement. De ce fait, cette partition sera formatée en EXT2.
- Une deuxième partition **/dev/nftla2** contenant le système de fichiers. Elle sera formatée en CRAMFS comme décrit plus loin.
- Une troisième partition **/dev/nftla2** contenant les quelques fichiers de configuration. Elle est de très petite taille et utilise en première approche un formatage EXT2.

Le fichier **/etc/fstab** décrivant les montages sur le système cible aura l'allure suivante :

```
bash-2.05# cat /etc/fstab
/dev/nftla2     /                       cramfs  defaults        1       1
/dev/nftla3     /data                   ext2    defaults        0       0
none            /dev/pts                devpts  mode=622        0       0
none            /proc                   proc    noauto          0       0
/dev/ram0       /var                    ext2    defaults        0       0
```

On note que la partition **/boot** n'est pas montée mais que nous avons une partition **/var** montée sur un disque mémoire **/dev/ram0**. Pour la deuxième partition (système de

fichiers principal), nous utiliserons le système de fichiers CRAMFS. Pour utiliser CRAMFS, il faut disposer du programme **mkcramfs**. La version livrée avec certaines distributions ne fonctionne pas forcément très bien et il vaut mieux récupérer la version officielle disponible sur le site du projet, soit *http://sourceforge.net/projects/cramfs*.

Pour construire une partition CRAMFS sur un support physique on devra tout d'abord créer l'image du répertoire au format CRAMFS.

```
# mkcramfs my_root my_root.img
Directory data: 14092 bytes
Everything: 4252 kilobytes
Super block: 76 bytes
CRC: 8647fdf8
```

On pourra ensuite copier l'image sur la partition par un simple **dd** ou un **cp** :

```
# dd < my_root.img > /dev/nftla2
```

Au niveau du noyau, il faudra bien entendu valider le support de CRAMFS en statique au niveau du menu File systems de la configuration du noyau. Puisque nous utilisons également un disque mémoire, il faudra valider le support Ramdisk le menu Block devices. La taille par défaut de 4 Mo pour le disque mémoire est suffisante pour notre application.

Si nous faisons référence au chapitre 4, nous savons que le démarrage du système est divisé en 5 étapes :

- le démarrage du système par LILO (Linux LOader) ou un programme équivalent type GRUB ;
- le chargement du noyau ;
- le lancement par le noyau du processus **init** (soit **/sbin/init**) ;
- lecture du fichier /etc/inittab par le processus **init**. Ce fichier contenant le nom du fichier de démarrage comme décrit ci-dessous.

```
# System initialization (runs when system boots).
  si:S:sysinit:/etc/rc.d/rc.S
```

- l'exécution du script ci-dessus.

Sachant que le répertoire **/var** est placé dans un disque mémoire, il est nécessaire de construire l'arborescence de ce dernier à chaque démarrage du système. On aura donc dans le fichier **rc.S** les lignes suivantes :

```
# Create /var (EXT2 filesystem)
/sbin/mke2fs -vm0 /dev/ram0 4096

# mount file systems in fstab (and create an entry for /)
# Mount /proc first to avoid warning about /etc/mtab
mount /proc
/bin/mount -at nonfs
```

```
# Populate /var
mkdir -p /var/tmp /var/log /var/lock /var/run /var/spool /var/lib/dhcp /var/run/dhcpc
mkdir -p /var/cron/tabs /var/www/html /var/spool/cron /var/spool/mail /var/ppp
chmod a+rwx /var/tmp
...
```

Outre la création de **/var**, on notera le lien symbolique de **/etc/mtab** vers **/proc/mounts**. Ce lien est nécessaire car **/etc** n'est pas accessible en écriture.

```
lrwxrwxrwx    1 root     root  12 Jun 23  2003 mtab -> /proc/mounts
```

De même, on notera l'utilisation fréquente des liens symboliques afin de permettre à des fichiers créés au démarrage d'apparaître dans le système de fichiers en lecture simple (voir exemple de **/etc/resolv.conf** ci-dessous). Les fichiers variables sont systématiquement placés dans le répertoire **/var/run** :

```
# ls -l /etc/resolv.conf
lrwxrwxrwx   1 root    root     20 Sep 24 07:53 /etc/resolv.conf -> /var/run/resolv.conf
```

Concernant les fichiers de configuration, ils sont placés dans le répertoire **/data** associé à la troisième partition :

```
# ls -l /data/sysconfig/
total 6
-rw-r--r--    1 65534    nobody         41 Jun 23  2003 general
-rw-r--r--    1 65534    nobody        349 Sep 17 21:14 network
-rw-r--r--    1 root     root          174 Jun 23  2003 ppp
-rw-r--r--    1 root     root          150 Sep  6  2003 pppoe
-rw-r--r--    1 root     root           16 Jun 23  2003 syslogd.opts
-rw-r--r--    1 root     root          150 Jun 24  2003 wireless
```

Mise au point des programmes

Utilisation de GDB

La mise au point des programmes ou « débogage » est un mal nécessaire connu de tous les développeurs. Lorsqu'on travaille sur une station Linux, la tâche est facilitée par l'utilisation du débogueur symbolique GDB (*GNU DeBugger*) qui permet d'exécuter le programme en pas à pas, de gérer des points d'arrêt, ou pire encore.

Si le système cible utilise une version très proche de celle du poste de développement, nous essaierons la plupart du temps de régler les problèmes directement sur la station de développement en s'approchant au maximum de l'environnement de la cible. Dans le cas où des problèmes inhérents à l'environnement cible subsistent, GDB offre la possibilité d'effectuer un débogage à distance (*remote debugging*) à travers un lien RS-232 ou une connexion réseau TCP/IP. Même si ce n'est pas directement lié au sujet de l'ouvrage, nous allons brièvement rappeler quelques concepts généraux concernant l'utilisation de GDB.

Si nous considérons le petit programme C qui suit :

```
#include <stdio.h>
#include <stdlib.h>

void affiche (int v)
{
     printf ("valeur= %d\n", v);
}

main (int ac, char **av)
{
     int i;
     for (i = 0 ; i < 5 ; i++)
          affiche (i);
}
```

Ce dernier pourra être compilé sous Linux avec la commande :

```
$ gcc -g -o affiche affiche.c
```

Puis exécuté par :

```
$ ./affiche
valeur= 0
valeur= 1
valeur= 2
valeur= 3
valeur= 4
```

L'option **-g** indique au compilateur d'ajouter les informations de débogage utilisables par GDB. Si nous lançons la même exécution sous la commande **gdb**, nous obtenons :

```
$ gdb affiche
GNU gdb Red Hat Linux 7.x (5.0rh-15) (MI_OUT)
Copyright 2001 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb)
```

Si nous posons un point d'arrêt sur la fonction **affiche()**, et que nous démarrons le programme, nous obtenons le résultat suivant :

```
(gdb) b affiche
Breakpoint 1 at 0x8048466: file affiche.c, line 6.
(gdb) r
Starting program: /home/pierre/tmp/affiche

Breakpoint 1, affiche (v=0) at affiche.c:6
6            printf ("valeur= %d\n", v);
(gdb) c
```

```
Continuing.
valeur= 0

Breakpoint 1, affiche (v=1) at affiche.c:6
6            printf ("valeur= %d\n", v);
(gdb) c
Continuing.
valeur= 1

Breakpoint 1, affiche (v=2) at affiche.c:6
6            printf ("valeur= %d\n", v);
(gdb)
```

Ce type de manipulation très classique ne présente aucune difficulté dans un environnement de station de développement Linux. Supposons à présent que l'on veut mettre au point un programme exécuté sur une machine cible nommée *portable* depuis un poste de développement nommé *duron*. Nous partons du principe que les deux machines sont connectées au même réseau local TCP/IP, mais elles pourraient également être reliées par un simple câble série RS-232.

Pour mettre au point le programme, nous devons bien évidemment disposer du programme **gdb** sur le poste de développement, mais également d'un programme « serveur » sur le poste cible. Ce programme est nommé **gdbserver** et fait partie de la distribution des sources **gdb**. En revanche, il n'est en général pas distribué dans le paquetage RPM binaire car son utilisation est réservée à un usage très particulier. Pour information, ce programme est livré en standard dans les distributions Linux embarquées spécialisées, comme Lynux-Works BlueCat ou Lineo Embedix. Le couple **gdb/gdbserver** peut bien entendu être utilisé sur des architectures différentes si celles-ci sont supportées par **gdb** et **gdbserver**.

Pour compiler **gdbserver**, nous pouvons partir du paquetage RPM source de **gdb** disponible sur *ftp://ftp.redhat.com/redhat/redhat-7.2-en/os/i386/SRPMS/gdb-5.0rh-15.src.rpm*. Nous pouvons également partir des sources de **gdb** disponibles sur *ftp://ftp.gnu.org/pub/gnu/gdb*.

Comme décrit dans ce chapitre, nous pouvons installer le paquetage source au moyen de :

```
rpm -ivh gdb-5.0rh-15.src.rpm
```

puis appliquer les « patches » propres à la Red Hat 7.2 par la commande :

```
rpm -bp /usr/src/redhat/gdb.spec
```

Nous devons ensuite nous positionner sur le répertoire des sources, puis générer le fichier Makefile.

```
cd /usr/src/redhat/BUILD/gdb+dejagnu-20010813/gdb
./configure
```

puis compiler et installer le programme **gdbserver**.

```
cd gdb/gdbserver
make
make install
```

Sur le poste de développement *duron*, nous disposons du source du programme **affiche.c** et de l'exécutable **affiche** compilé avec l'option **-g**.

```
[pierre@duron pierre]$ ls -l
total 32
-rwxr-xr-x    1 pierre   pierre      24622 mai  2 08:19 affiche
-rw-r--r--    1 pierre   pierre        194 mai  2 08:21 affiche.c
```

Sur la machine cible, nous devons disposer d'une copie de l'exécutable **affiche**. Côté cible, nous devons tout d'abord lancer **gdbserver** sur le programme **affiche**.

```
[pierre@portable tmp]$ gdbserver duron:2345 affiche
Process affiche created; pid = 12810
```

Nous choisissons d'utiliser un lien réseau entre les deux machines. Le premier paramètre de **gdbserver** est le nom réseau (ou l'adresse IP) du poste de développement associé au numéro de port TCP à utiliser, ici 2345.

Attention

Nous rappelons que les ports inférieurs à 1024 ne doivent pas être utilisés, car réservés au système. De même, il faut prendre soin de choisir un port non utilisé par une autre application utilisateur.

Côté poste de développement, il suffit de lancer **gdb** avec le nom du programme en paramètre. La sélection de la cible à déboguer s'effectue grâce à la commande **target**.

```
$ gdb affiche
GNU gdb Red Hat Linux 7.x (5.0rh-15) (MI_OUT)
Copyright 2001 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) target remote portable:2345
Remote debugging using portable:2345
0x40001e60 in ?? ()
(gdb)
```

Côté cible, la ligne suivante est affichée :

```
Remote debugging using duron:2345
```

À partir de là, on peut utiliser **gdb** de manière classique en posant par exemple un point d'arrêt sur la fonction **affiche()**.

```
(gdb) b affiche
Breakpoint 1 at 0x8048466: file affiche.c, line 6.
(gdb) c
Continuing.

Breakpoint 1, affiche (v=0) at affiche.c:6
```

```
warning: Source file is more recent than executable.

6            printf ("valeur= %d\n", v);
(gdb) n
7       }
(gdb)  c
Continuing.

Breakpoint 1, affiche (v=1) at affiche.c:6
6            printf ("valeur= %d\n", v);
(gdb) n
7       }
```

Côté cible, on obtient l'affichage de l'exécution du programme.

```
valeur= 0
valeur= 1
```

Si l'on inhibe le point d'arrêt et que l'on finit l'exécution du programme, l'exécution se termine côté cible et **gdbserver** s'arrête. On obtient côté poste de développement :

```
(gdb) dis 1
(gdb) c
Continuing.

Program exited with code 0224.
(gdb)
```

et côté cible :

```
valeur= 2
valeur= 3
valeur= 4

Child exited with retcode = 94

Child exited with status 0
GDBserver exiting
[pierre@portable tmp]$
```

On pourra de la même manière utiliser une connexion par un câble série RS-232 sur le port COM1 (**/dev/ttyS0**) en lançant côté cible :

```
gdbserver /dev/ttyS0 nom_programme
```

et côté système de développement :

```
gdb nom_programme
```

puis dans **gdb** :

```
(gdb) target remote /dev/ttyS0
```

Si l'on veut utiliser une autre vitesse que 9600 bits/s, on devra préciser l'option **--baud** à **gdb**.

Utilisation de `strace`

La commande **strace** permet de visualiser les appels systèmes et les signaux utilisés dans un programme. Le résultat de **strace** est assez verbeux comme en témoigne le test effectué sur notre petit programme **affiche**. La sortie de **strace** peut cependant être paramétrée en utilisant l'option **-e**.

```
$ strace ./affiche
execve("./affiche", ["./affiche"], [/* 22 vars */]) = 0
uname({sys="Linux", node="duron.localdomain", ...}) = 0
brk(0)                                  = 0x804964c
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=38627, ...}) = 0
old_mmap(NULL, 38627, PROT_READ, MAP_PRIVATE, 3, 0) = 0x40017000
close(3)                                = 0
open("/lib/i686/libc.so.6", O_RDONLY)   = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0 \306\1"..., 1024) = 1024
fstat64(3, {st_mode=S_IFREG|0755, st_size=5772268, ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0
➥ x40021000
old_mmap(NULL, 1290088, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) = 0x40022000
mprotect(0x40154000, 36712, PROT_NONE)  = 0
old_mmap(0x40154000, 20480, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 3, 0x13
➥ 1000) = 0x40154000
old_mmap(0x40159000, 16232, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANON
➥ YMOUS, -1, 0) = 0x40159000
close(3)                                = 0
munmap(0x40017000, 38627)               = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40
➥ 017000
write(1, "valeur= 0\n", 10valeur= 0
)             = 10
write(1, "valeur= 1\n", 10valeur= 1
)             = 10
write(1, "valeur= 2\n", 10valeur= 2
)             = 10
write(1, "valeur= 3\n", 10valeur= 3
)             = 10
write(1, "valeur= 4\n", 10valeur= 4
)             = 10
munmap(0x40017000, 4096)                = 0
_exit(-1073743148)                      = ?
```

La commande **strace** peut être très utile dans le cas de la mise au point d'un pilote de périphérique.

Détection des problèmes de mémoire

Le problème de dépassement de mémoire allouée est la bête noire de tous les développeurs C/C++. En général, cela tient à ce qu'un programme utilise de la mémoire qu'il n'a pas préalablement allouée. Le cas typique est le dépassement de tableau, lorsqu'on alloue un certain nombre d'éléments d'un tableau dynamique avec l'appel système **`malloc()`** et que l'on utilise ensuite plus d'éléments que le nombre prévu.

Le problème est d'autant plus épineux qu'il n'apparaît pas forcément immédiatement au cours de l'exécution du programme car il a trait à la mémoire disponible, et donc à la charge du système. Dans le cas d'un système embarqué, les conséquences peuvent être dramatiques car le défaut peut apparaître bien après la phase de développement.

Le petit programme C présenté ci-après alloue par exemple 10 cellules d'un tableau de caractères, mais en utilise 20.

```
#include <stdio.h>
#include <stdlib.h>

main (int ac, char **av)
{
  register int i;
  char *tab = calloc (1, 10);

  for (i = 0 ; i < 20 ; i++)
    *(tab+i) = i;

  for (i = 0 ; i < 20 ; i++)
    printf ("tab[%d]= %d\n", i, *(tab+i));
}
```

Après compilation et exécution, le système n'y voit pourtant que du feu.

```
$ gcc -g -o buggy buggy.c
$ ./buggy
tab[0]= 0
tab[1]= 1
tab[2]= 2
tab[3]= 3
...
tab[18]= 18
tab[19]= 19
```

Pour détecter ce type de problème, nous devons utiliser un allocateur de mémoire spécial. Les distributions classiques fournissent le paquetage ElectricFence (la « barrière électrifiée »), qui remplace l'allocateur standard par une version de débogage. Pour cela, il faut installer le paquetage associé.

```
rpm -ivh ElectricFence-2.2.2-8.i386.rpm
```

Il suffit ensuite de compiler le programme en utilisant la bibliothèque **libefence**. Une nouvelle exécution du programme détecte bien un problème de mémoire et le programme est interrompu avec une erreur de segmentation (signal **SIGSEGV**).

```
$ gcc -g -o buggy buggy.c -lefence
$ ./buggy
  Electric Fence 2.2.0 Copyright (C) 1987-1999 Bruce Perens <bruce@perens.com>
Segmentation fault
```

Le système doit dès lors générer un fichier **core** qui permet de localiser l'erreur. Si le fichier **core** n'est pas présent, il faut vérifier la configuration de la session grâce à la commande **ulimit -c**. Si la valeur est 0, il faut la modifier en donnant la nouvelle valeur « unlimited ». Une nouvelle exécution du programme générera cette fois-ci un fichier **core** (message *core dumped*).

```
$ ulimit -c
0
$ ulimit -c unlimited
$ ulimit -c
unlimited
$ ./buggy
  Electric Fence 2.2.0 Copyright (C) 1987-1999 Bruce Perens <bruce@perens.com>
Segmentation fault (core dumped)
```

Le débogueur **gdb** permet ensuite de localiser l'erreur.

```
$ gdb buggy core
GNU gdb Red Hat Linux 7.x (5.0rh-15) (MI_OUT)
Copyright 2001 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
Core was generated by `./buggy'.
Program terminated with signal 11, Segmentation fault.
Reading symbols from /usr/lib/libefence.so.0...done.
Loaded symbols for /usr/lib/libefence.so.0
Reading symbols from /lib/i686/libc.so.6...done.
Loaded symbols for /lib/i686/libc.so.6
Reading symbols from /lib/i686/libpthread.so.0...done.

warning: Unable to set global thread event mask: generic error
[New Thread 1024 (LWP 1671)]
Error while reading shared library symbols:
Cannot enable thread event reporting for Thread 1024 (LWP 1671): generic error
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
#0  0x080485b5 in main (ac=1, av=0xbffffb64) at buggy.c:11
11          *(tab+i) = i;
(gdb)
```

On peut également exécuter le programme **buggy** directement dans gdb, ce qui conduit au même résultat.

```
$ gdb buggy
GNU gdb Red Hat Linux 7.x (5.0rh-15) (MI_OUT)
Copyright 2001 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) r
Starting program: /home/pierre/buggy
[New Thread 1024 (LWP 1675)]

  Electric Fence 2.2.0 Copyright (C) 1987-1999 Bruce Perens <bruce@perens.com>

Program received signal SIGSEGV, Segmentation fault.
[Switching to Thread 1024 (LWP 1675)]
0x080485b5 in main (ac=1, av=0xbffffb54) at buggy.c:11
11          *(tab+i) = i;
(gdb)
```

La bibliothèque **efence** permet de détecter les débordements de mémoires mais certains produits commerciaux, beaucoup plus complexes (et plus coûteux), ont d'autres fonctionnalités intéressantes comme la détection de fuites de mémoire. Nous pouvons citer le produit Insure++ édité par Parasoft (*http://www.parasoft.com*).

En résumé

Le présent chapitre a décrit quelques fonctions et méthodes très utiles pour le développement d'un système Linux embarqué. Concernant les disques et mémoires flash, les produits M-Systems sont très répandus et peuvent être utilisés soit avec des pilotes fournis par le constructeur, soit *via* la couche MTD intégrée dans le noyau Linux 2.4. Le pilote M-Systems utilisant une bibliothèque non-GPL, le fait de redistribuer un noyau Linux contenant ce pilote en statique n'est pas conforme à la GPL. On pourra alors utiliser un disque mémoire initial (**initrd**) contenant les modules M-Systems ou bien utiliser le pilote MTD.

La couche MTD permet également de piloter d'autres types de mémoires flash comme les produits compatibles CFI.

Différents formats de système de fichiers incluant des fonctionnalités de journalisation et/ou de compression en temps réel sont utilisables dans un environnement Linux embarqué.

8

Autres techniques de démarrage : Loadlin, LinuxBIOS, RedBoot

Un autre système de démarrage : LOADLIN

LOADLIN est un programme permettant de charger un noyau Linux depuis un système de type MS-DOS. Le principal intérêt, outre le fait de ne pas avoir à modifier le secteur de démarrage du système, en est de pouvoir charger Linux après avoir initialisé le matériel dans un environnement DOS.

La syntaxe de chargement du noyau est la suivante :

```
c:\loadlin\loadlin.exe c:\linux\vmlinuz root=/dev/hda3 ro
```

qui signifie que l'image **vmlinuz** est chargée depuis DOS et utilise la partition **/dev/hda3** comme partition principale, montée initialement en lecture seule.

LOADLIN est également utilisé dans certains systèmes embarqués de petite taille initialement destinés à l'environnement DOS. La carte d'évaluation DIL décrite au chapitre 3 et diffusée par SSV (*http://www.ssv-embedded.de*) utilise ce principe, que nous allons décrire en tant qu'illustration fonctionnelle de **LOADLIN** et des disques mémoire. Comme décrit au chapitre 3, le système est constitué d'un processeur AMD compatible 486, de 8 Mo de RAM et 2 Mo de mémoire flash. Le test présenté ici est réalisé sur une carte d'évaluation permettant le dialogue *via* un port série ou un lien Ethernet comme décrit dans la figure ci-après :

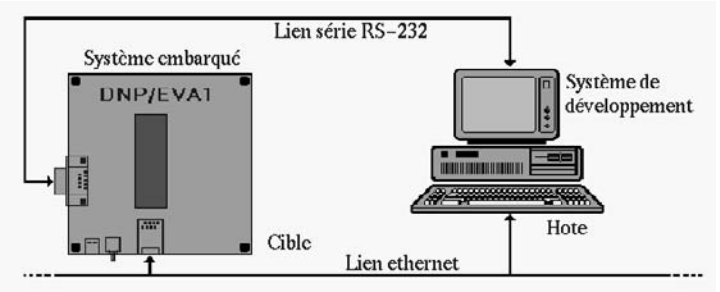


Figure 8-1
Connexions DIL/NetPC.

Le poste de développement est équipé d'un émulateur de terminal comme **`minicom`**, fourni en standard sur les distributions Linux. Au démarrage du DIL/NetPC, on obtient l'affichage suivant :

```
ESB486 for DIL/NetPC DNP/1486-3V  V.0.15
Copyright 2000 SSV SOFTWARE SYSTEMS GmbH

  TESTING INTERVAL TIMER........... PASS
  TESTING INTERRUPT CONTROLLER..... PASS
  TESTING DMA CONTROLLER........... PASS
  TESTING SYSTEM MEMORY........640K PASS
  TESTING EXTENDED MEMORY.....7168K PASS
  TESTING REAL TIME CLOCK.......... PASS
  TESTING CMOS RAM CHECKSUM........ PASS

FlashFx 4.02.204 (386 DOS)
Copyright (c) 1993-1999, Datalight Inc.
Datalight Patent US#5860082

Starting ROM-DOS...

HIMEM v7.10 (Revision 3.00.44)
Copyright (c) 1989-2000 Datalight, Inc.
Using PS2 A20 Control (ON)
32 XMS handles available.
Minimum HMA usage is OK.

VDISK v6.22 (Revision 3.00.44)
Copyright (c) 1989-2000 Datalight, Inc.

Installed 4096KB XMS RAM disk as drive D:
C:>
```

La flash contient une version réduite d'un système compatible MS-DOS.

```
C:\>dir

 Volume in drive C is SSD
 Volume Serial Number is 133C-13F0
 Directory of C:\

COMMAND  COM        34,685 05-02-2000  12:30p
AUTOEXEC BAT            32 03-06-2000   6:05p
CONFIG   SYS            74 05-02-2000  12:12p
HIMEM    SYS         5,833 05-02-2000  12:29p
RB       COM         3,095 05-02-2000  12:54p
SB       COM         2,997 05-02-2000  12:54p
VDISK    SYS         8,092 05-02-2000  12:29p
LINUX        <DIR>         01-01-1980  12:00a
        8 file(s)         54,808 bytes
                         176,640 bytes free
```

Le système fournit deux programmes **RB.COM** et **SB.COM** permettant respectivement de recevoir et d'envoyer des fichiers *via* le port série en utilisant le protocole Y-MODEM. Le répertoire Linux contient une arborescence similaire à celle décrite à la section précédente.

```
C:\>cd linux
C:\LINUX>dir

 Volume in drive C is SSD
 Volume Serial Number is 133C-13F0
 Directory of C:\LINUX

.            <DIR>         01-01-1980  12:00a
..           <DIR>         01-01-1980  12:00a
LOADLIN  EXE        10,819 01-01-1980  12:05a
RIMAGE   GZ        996,972 01-01-1980  12:36a
ZIMAGE             418,978 01-01-1980  12:10a
START    BAT           161 01-01-1980  12:11a
        6 file(s)      1,426,930 bytes
                         176,640 bytes free
```

Le fichier **ZIMAGE** constitue le noyau Linux qui utilise le fichier **RIMAGE.GZ** comme image compressée de disque mémoire. Le script **START.BAT** permet de démarrer le noyau Linux grâce à l'utilitaire **LOADLIN.EXE**.

```
C:\LINUX>type start.bat
@echo off
ECHO Start Embedded Linux for DilNet ...
loadlin zimage console=ttyS0,115200 initrd=rimage.gz root=/dev/ram
ECHO Embedded Linux stopped or failed!
```

Vu que Linux est chargé uniquement en mémoire vive, toutes les modifications effectuées après le démarrage sont perdues à l'arrêt du système. Les modifications permanentes doivent être sauvegardées dans le fichier **RIMAGE.GZ** Après lancement de la commande **START**, on obtient l'écran suivant :

```
Linux version 2.2.5 (root@mha) (gcc version egcs-2.91.66 19990314 (egcs-1.1.2
➥ release)) #11 Fri Jun 30 11:09:38 MEST 2000
Console: colour *CGA 80x25
Calibrating delay loop... 16.54 BogoMIPS
Memory: 5740k/8192k available (748k kernel code, 412k reserved, 288k data, 28k init)
Checking if this processor honours the WP bit even in supervisor mode... Ok.
CPU: AMD 02/0a stepping 04
Checking 'hlt' instruction... OK.
Posix conformance testing by UNIFIX
Linux NET4.0 for Linux 2.2
Based upon Swansea University Computer Society NET3.039
NET4: Unix domain sockets 1.0 for Linux NET4.0.
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
Starting kswapd v 1.5
Serial driver version 4.27 with no serial options enabled
ttyS00 at 0x03f8 (irq = 4) is a 16550A
Keyboard timeout[2]
Keyboard timeout[2]
RAM disk driver initialized:  16 RAM disks of 20480K size
loop: registered device at major 7
RAMDISK: Compressed image found at block 0
Uncompressing.............................................done.
VFS: Mounted root (minix filesystem).
Freeing unused kernel memory: 28k freed
eth0: cs8900 rev J found at 0x300 media RJ-45, IRQ 5 02 80 ad 20 26 b6 ;-)
eth0: using 10Base-T (RJ-45)

SSV DNPX Linux - Version 0.04
emblinux login:
```

Un compte *gast* est disponible pour accéder au système. Ce dernier est également disponible en accès Telnet et FTP.

```
emblinux login: gast
Password:
# ls -l
# pwd
/home/gast
#
```

Nous allons maintenant effectuer une modification de l'image **RIMAGE.GZ** en y ajoutant un petit programme de test sur **/home/gast**. Pour cela, il faut déjà récupérer le fichier **RIMAGE.GZ** sur le poste de développement en utilisant la commande **SB.COM** sous DOS.

```
C:\LINUX>..\SB RIMAGE.GZ

SB.COM Send Y-ModemG (Batch) Version 0.17 (c) SSV 2000
Ready to send file(s) 'RIMAGE.GZ'. End with CTRL-C!
```

Côté poste de développement, on utilise la fonction de réception X/Y/Z-MODEM de **minicom** qui est activée en saisissant *Control-X R*. Lorsque le fichier est sur le poste de développement, il faut le décompresser puis le monter en utilisant le device *loopback*.

```
# gunzip RIMAGE.GZ
# mount -t minix -o loop RIMAGE /mnt/loop0
# ls -l /mnt/loop0
total 17
drwxr-xr-x    2 root     root         2144 jun 11  2001 bin
drwxr-xr-x    2 root     root           64 jun 28  2000 boot
drwxr-xr-x    2 root     root         1984 jui  4  2000 dev
drwxr-xr-x   14 root     root         1376 avr 30 09:13 etc
drwxr-xr-x    3 root     root           96 jun  7  2000 home
drwxr-xr-x    3 root     root          416 jui 10  2000 lib
drwxr-xr-x    2 root     root           64 mar  3  1999 mnt
drwxr-xr-x    2 root     root           64 fév  9  2000 proc
drwxrwx---    2 root     root           64 jun 15  2000 root
drwxr-xr-x    2 root     root          864 jui  3  2000 sbin
drwxrwxrwt    2 root     root           64 jun 15  2000 tmp
drwxr-xr-x    6 root     root          192 mai 25  2000 usr
drwxr-xr-x    7 root     root          288 mar 20  1998 var
```

Notez que cette image utilise le système de fichier *minix* (premier système de fichier utilisé par Linux) et non le système de fichier ext2. Cela ne change rien à la suite.

On crée ensuite un programme minimal en C, que l'on compile en utilisant l'option **-s** afin de réduire la taille de l'exécutable. Le fichier **hello** est ensuite copié dans le répertoire **/mnt/loop0/home/gast**.

```
# cat hello.c
main ()
{
        printf ("hello world\n");
}
# cc -s -o hello hello.c
# ls -l hello
-rwxr-xr-x    1 root     root         3012 avr 30 10:03 hello
# ./hello
hello world
# cp hello /mnt/loop0/home/gast
```

On démonte ensuite le système et l'on compresse de nouveau le fichier **RIMAGE**.

```
# umount /mnt/loop0
# gzip -9 RIMAGE
```

Il reste à transférer la nouvelle image compressée sur la cible en utilisant la commande **RB.COM** et la séquence *Control-X S* côté **minicom**.

```
C:\LINUX>..\RB
RB.COM Receive Y-ModemG (Batch) Version 0.17 (c) SSV 2000
Wait for Files. End with CTRL-C!
```

Il reste à démarrer de nouveau Linux et à tester le programme `hello`.

```
emblinux login: gast
Password:
# ls -l
-rwxr-xr-x    1 root     root        3012 Apr 30  2002 hello
# ./hello
hello world
```

LinuxBIOS

Le projet LinuxBIOS (*http://www.linuxbios.org*) vise à remplacer le BIOS par un noyau Linux modifié sur des systèmes de type x86 et Alpha. Comme la majorité des systèmes d'exploitation modernes, Linux n'utilise pas le BIOS et sa présence est plus une contrainte qu'un avantage. L'utilisation de LinuxBIOS à la place du BIOS traditionnel est avantageuse sur plusieurs points :

- Au niveau des performances, car le temps de démarrage du BIOS est supprimé et il ne reste que le temps de boot du noyau Linux.
- Au niveau de la sécurité, car le BIOS est désormais un vrai noyau Linux avec un administrateur unique.
- Au niveau de la maintenance, car celle-ci peut être faite à distance. De même, le système peut être contrôlé depuis une console série ou un accès réseau.

LinuxBIOS est d'ores et déjà disponible pour plusieurs *chipsets* Intel, compatibles et Alpha. Une page d'information maintient la liste des chipsets supportés par LinuxBIOS sur *http://www.acl.lanl.gov/linuxbios/status*.

Plusieurs produits commerciaux utilisent également LinuxBIOS (voir *http://www.acl.lanlgov/linuxbios/products)*. La société CWLINUX propose en particulier le LinuxBIOS SDK, constitué d'une carte mère compatible LinuxBIOS (chipset SiS 630/730) et des outils de développement logiciel associés (voir *http://www.cwlinux.com/eng/products)*.

RedBoot

RedBoot est l'acronyme de *Red Hat Embedded Debug and Bootstrap*. RedBoot est dérivé du développement d'eCos (*Embeddable Configurable Operating System*), un système d'exploitation temps réel à très faible empreinte mémoire développé par Red Hat. RedBoot apporte des fonctions intéressantes concernant des points comme le démarrage d'un système *via* réseau (bootp et tftpboot), la gestion de la mémoire flash, le téléchargement de fichier *via* réseau ou ligne série, l'exécution d'un noyau Linux sur la cible ou bien la mise au point à distance.

Présentation et principales commandes

RedBoot est disponible pour un grand nombre d'architectures comme les processeurs StrongARM, SuperH ou x86. Le site de référence de RedBoot est situé à l'adresse *http://*

sources.redhat.com/redboot. Une liste d'images binaires précompilées est disponible sur *http://sources.redhat.com/eCos/boards/redbootbins*.

L'interface d'utilisation de RedBoot est relativement simple à utiliser. La documentation complète est disponible en ligne à l'adresse *http://sources.redhat.com/eCos/docs-latest/redboot/redboot.html* (ou bien *redboot.pdf* pour la version PDF).

La commande **fconfig** permet de connaître et modifier la configuration courante de RedBoot.

```
RedBoot> fconfig -l
 Run script at boot: false
 Use BOOTP for network configuration: false
 Local IP address: 192.168.3.10
 Default server IP address: 192.168.3.1
 GDB connection port: 9000
 Network debug at boot time: false
```

On peut communiquer avec RedBoot en utilisant le protocole Telnet sur le port 9000.

```
$ telnet mon_redboot 9000
Connected to mon_redboot
Escape character is `^]'.
RedBoot>
```

La configuration réseau de RedBoot s'effectue soit en spécifiant une adresse IP statique, soit en affectant une adresse IP dynamique *via* le protocole BOOTP. Cela nécessite la mise en place d'un serveur DHCP sur le poste de développement. Il faut noter que RedBoot ne supporte par le routage IP et que le poste de développement devra être sur le même réseau que la cible supportant RedBoot.

RedBoot permet de manipuler de la mémoire flash si elle est disponible sur le système cible, et ce *via* le FIS (*Flash Image System*) et la commande **fis**.

```
RedBoot> fis init -f
About to initialize [format] flash image system - are you sure (y/n)? n
```

La commande **fis** permet également la manipulation de partitions sur la mémoire flash (**fis create**, **fis delete**, **fis list**) ou le chargement de la mémoire flash vers la RAM (**fis load**).

La commande **load** permet de télécharger un fichier dans la RAM depuis le poste de développement *via* le réseau ou une connexion série. Dans le cas d'une connexion réseau, le protocole TFTP est utilisé. En cas de lien série, on peut utiliser XMODEM ou YMODEM. Le fichier est téléchargé à une adresse de la mémoire RAM de la cible en utilisant l'option **-b**.

```
RedBoot> load -b -r 0x8x210000 -m TFTP -h 192.168.3.1 bzImage
```

La commande **exec** permet d'exécuter un noyau Linux depuis la RAM.

Un exemple complet d'utilisation

La section qui suit présente un exemple complet d'installation et d'utilisation de RedBoot sur une carte d'évaluation Assabet StrongARM SA1110 fournie par Intel. L'environnement testé est le suivant.

Du côté matériel, l'expérience requiert :

- une carte Intel Assabet qui est une carte d'évaluation du processeur StrongARM SA1110 (voir *http://developer.intel.com/design/pca/applicationsprocessors/1110_brf.htm* et *http://developer.intel.com/design/pca/applicationsprocessors*),
- connexion JTAG (*Joint Test Advisory Group*) sur le port parallèle,
- connexion série.

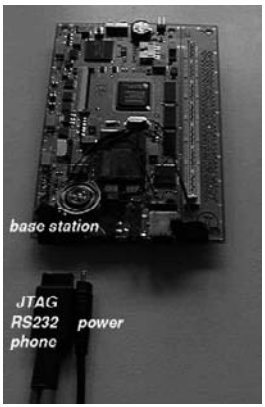


Figure 8-2
Carte Assebet Intel.

Remarque

La norme JTAG ou IEEE 1149.1 permet de tester les parties numériques des systèmes, cartes électroniques et ASIC. Des informations en français sur le JTAG sont disponibles sur *http://www.temento.com/ftestintr.htm*.

Côté logiciel, nous utilisons les composants suivants :

- noyau Linux 2.4.18 avec patch ARM (*http://www.arm.linux.org.uk*),
- distribution Familiar (disponible sur *http://www.handhelds.org*),
- RedBoot.

L'expérience est composée des phases suivantes.

1. Récupération, configuration et compilation de RedBoot.
2. « Flashage » de RedBoot sur la carte grâce au JTAG *via* le port parallèle.
3. Prise de contrôle de la carte *via* le port série.
4. Configuration de RedBoot sur la carte, partitionnement de la mémoire flash et configuration réseau de RedBoot.
5. Récupération du noyau Linux *via* le protocole TFTP et flashage.
6. Récupération du système de fichier *via* TFTP, puis flashage.
7. Démarrage.

Remarque

Le terme « flasher » ou « flashage » ne figure certainement pas dans le dictionnaire de l'Académie française. Il est cependant bien approprié pour exprimer simplement une « écriture dans une mémoire permanente ».

Une documentation est disponible dans les sources du noyau dans le fichier `Documentation/arm/SA1100/Assabet.txt`. Concernant RedBoot, la documentation est disponible en ligne sur *http://sources.redhat.com/eCos/docs-latest/redboot/redboot.html*.

Il est avant tout nécessaire de récupérer eCos de Red Hat car RedBoot est intégré à ce projet. Mieux vaut télécharger la dernière version à partir de l'arborescence CVS. Les instructions de téléchargement sont disponibles sur *http://sources.redhat.com/eCos/anoncvs.html*.

Après téléchargement de eCos et extraction dans le répertoire **eCos**, un répertoire **redboot_assabet** est créé au même niveau que le répertoire **eCos** afin d'accueillir notre configuration.

```
# ls
eCos  redboot_assabet
# export eCosDIR=/home/src/rh/eCos
# export eCos_REPOSITORTY=/home/src/rh/eCos/packages
# cd redboot_assabet
# $eCosDIR/host/configure --prefix=$TEMP/redboot-build --with-tcl-header=/usr/
  ➥include/tcl8.3/ --with-tcl-lib=/usr/lib/tcl8.3
# make
```

La configuration s'effectue ensuite comme suit :

```
# ./tools/configtool/standalone/common/eCosconfig new assabet redboot
U CYGSEM_HAL_USE_ROM_MONITOR, new inferred value 0
U CYGDBG_HAL_COMMON_CONTEXT_SAVE_MINIMUM, new inferred value 0
U CYGDBG_HAL_DEBUG_GDB_INCLUDE_STUBS, new inferred value 1
U CYGFUN_LIBC_STRING_BSD_FUNCS, new inferred value 0
# ./tools/configtool/standalone/common/eCosconfig
➥ import $eCosDIR/packages/hal/arm/sa11x0/assabet/current/misc/redboot_ROM.ecm
# ./tools/configtool/standalone/common/eCosconfig tree
# make
```

Les binaires sont alors disponibles dans le répertoire **install/bin**.

```
# ls -al install/bin
total 1157
drwxr-xr-x    2 root     root          176 jun 20 15:18 .
drwxr-xr-x    5 root     root          120 jun 20 15:18 ..
-rwxr-xr-x    1 root     root       103160 jun 20 15:18 redboot.bin
-rwxr-xr-x    1 root     root       681261 jun 20 15:18 redboot.elf
-rwxr-xr-x    1 root     root       193021 jun 20 15:18 redboot.img
-rwxr-xr-x    1 root     root       309612 jun 20 15:18 redboot.srec
```

Pour flasher l'utilitaire RedBoot sur la carte, on utilise le programme *Jflash* développé par Intel et adapté à Linux.

```
# Jflash-linux install/bin/redboot.bin
JFLASH Version 1.2
Using LPT port at 0x378
SA-1110 revision B4
Number of blocks in device = 128
Block size = 65536 0x10000
Device size = 16777216 0x1000000

Starting erase
Erasing done
Starting programming
Programming done
Starting verify
Verification successful!
```

Sur une session **minicom** connectée au port série, on obtient la trace suivante :

```
Waiting for network card: No network interfaces found

RedBoot(tm) bootstrap and debug environment [ROM]
Non-certified release, version UNKNOWN - built 15:18:20, Jun 20 2002

Platform: Assabet development system (StrongARM 1110)
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x02000000, 0x000106c8-0x01fbd000 available
FLASH: 0x50000000 - 0x52000000, 128 blocks of 0x00040000 bytes each.
RedBoot>
```

On doit ensuite initialiser la mémoire flash et configurer RedBoot au niveau du réseau.

```
RedBoot> fis init -f
About to initialize [format] FLASH image system - are you sure (y/n)? y
*** Initialize FLASH Image System
... Erase from 0x50080000-0x51f80000:
.....................................................................................
... Erase from 0x51fc0000-0x51fc0000:
... Erase from 0x52000000-0x52000000:
... Unlock from 0x51fc0000-0x52000000: .
... Erase from 0x51fc0000-0x52000000: .
... Program from 0x01fbf000-0x01fbf400 at 0x51fc0000: .
... Lock from 0x51fc0000-0x52000000: .
RedBoot> fconfig -i
Initialize non-volatile configuration - are you sure (y/n)? y
Run script at boot: false
Use BOOTP for network configuration: false
Local IP address: 10.0.0.2
Default server IP address: 10.0.0.1
Console baud rate: 115200
GDB connection port: 9000
Network debug at boot time: false
Update RedBoot non-volatile configuration - are you sure (y/n)? y
... Unlock from 0x51f80000-0x51f81000: .
... Erase from 0x51f80000-0x51f81000: .
... Program from 0x01fbe000-0x01fbf000 at 0x51f80000: .
... Lock from 0x51f80000-0x51f81000: .
RedBoot>
```

On insère ensuite l'adaptateur Ethernet (au format CompactFlash) puis on effectue une initialisation de la carte.

```
RedBoot> reset
... Resetting.ÿü.Failed
Socket Communications, Inc: Low Power Ethernet CF Revision C 5V/3.3V 08/27/98
Ethernet eth0: MAC address 00:c0:1b:00:b6:a1
IP: 10.0.0.2, Default server: 10.0.0.1
```

```
RedBoot(tm) bootstrap and debug environment [ROM]
Non-certified release, version UNKNOWN - built 15:18:20, Jun 20 2002

Platform: Assabet development system (StrongARM 1110)
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x02000000, 0x000106c8-0x01fbd000 available
FLASH: 0x50000000 - 0x52000000, 128 blocks of 0x00040000 bytes each.
RedBoot>
```

On peut ensuite télécharger puis flasher le noyau et l'image du système de fichiers. L'adresse IP 10.0.0.1 dispose d'un serveur FTP qui contient :

- le noyau sous forme du fichier **zI2418as**,
- l'image du système de fichier **task-familiar-complete.jffs2** (extrait de la distribution familiar disponible sur *http://www.handhelds.org*).

```
RedBoot> load zI2418as -r -b 0x100000
Raw file loaded 0x00100000-0x001b03a0
RedBoot> fis create "Linux kernel" -b 0x100000 -l 0xc0000
... Erase from 0x50040000-0x50100000: ...
... Program from 0x00100000-0x001c0000 at 0x50040000: ...
... Unlock from 0x51fc0000-0x52000000: .
... Erase from 0x51fc0000-0x52000000: .
... Program from 0x01fbf000-0x01fff000 at 0x51fc0000: .
... Lock from 0x51fc0000-0x52000000: .
RedBoot> load task-familiar-complete.jffs2 -r -b 0x100000
RedBoot> load fam.jffs2  -r -b 0x100000
Raw file loaded 0x00100000-0x00c80000
RedBoot> fis free
  0x50100000 .. 0x51F80000
RedBoot> fis unlock -f 0x50100000 -l 0x01E80000
... Unlock from 0x50100000-0x51f80000:
➥ .....................RedBoot> fis erase -f 0x50100000 -l 0x01E80000
... Erase from 0x50100000-0x51f80000:
➥ .....................RedBoot> fis write -b 0x100000 -l 0x00B80000 -f 0x50100000
* CAUTION * about to program FLASH
            at 0x50100000..0x50c7ffff from 0x00100000 - are you sure (y/n)? y
... Erase from 0x50100000-0x50c80000: ..........................................
... Program from 0x00100000-0x00c80000 at 0x50100000:
➥ .....................RedBoot> fis create "JFFS2" -n -f 0x50100000 -l 0x01E80000
... Unlock from 0x51fc0000-0x52000000: .
... Erase from 0x51fc0000-0x52000000: .
... Program from 0x01fbf000-0x01fff000 at 0x51fc0000: .
... Lock from 0x51fc0000-0x52000000: .
RedBoot>
```

La mémoire flash est maintenant initialisée ; elle contient le noyau et le système de fichiers.

```
RedBoot> fis list
```

```
Name              FLASH addr  Mem addr    Length      Entry point
RedBoot           0x50000000  0x50000000  0x00040000  0x00000000
RedBoot[backup]   0x50040000  0x50040000  0x00040000  0x00000000
RedBoot config    0x51F80000  0x51F80000  0x00001000  0x00000000
FIS directory     0x51FC0000  0x51FC0000  0x00040000  0x00000000
Linux kernel      0x50040000  0x00100000  0x000C0000  0x00000000
JFFS2             0x50100000  0x50100000  0x01E80000  0x00000000
RedBoot>
```

On peut enfin demander à RedBoot de charger le noyau depuis la mémoire flash vers la RAM, puis de l'exécuter.

```
RedBoot> fis load "Linux kernel"
RedBoot> exec -b 0x100000 -l 0xc0000 -c "root=/dev/mtdblock4 console=ttySA0"
Uncompressing Linux..................................... done, booting the kernel.
Linux version 2.4.18-rmk4 (root@ebenard.boudiow.org) (gcc version 2.95.2 19991024
➥ (release)) #96 ven avr 26 20:27:19 CEST 2002
Processor: Intel StrongARM-1110 revision 8
Architecture: Intel-Assabet
On node 0 totalpages: 4096
zone(0): 4096 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: root=/dev/mtdblock4 console=ttySA0
Warning: uninitialized Real Time Clock
Console: colour dummy device 80x30
Calibrating delay loop... 147.04 BogoMIPS
Memory: 16MB = 16MB total
Memory: 14396KB available (1343K code, 279K data, 64K init)
Dentry-cache hash table entries: 2048 (order: 2, 16384 bytes)
Inode-cache hash table entries: 1024 (order: 1, 8192 bytes)
Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
Buffer-cache hash table entries: 1024 (order: 0, 4096 bytes)
Page-cache hash table entries: 4096 (order: 2, 16384 bytes)
Posix conformance testing by UNIFIX
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
CPU clock: 221.200 MHz (0.000-221.200 MHz)
S4tarting kswapd
UCB1x00: probed IRQ44 correctly. Please remove machine dependencies from
➥ ucb1x00-core.c
ucb : pm_register
Sound: Assabet UDA1341: dsp id 3 mixer id 0
SA1100 flash: probing 32-bit flash bus
Using buffer write method
Using RedBoot partition definition
Creating 7 MTD partitions on "SA1100 flash":
0x00000000-0x00040000 : "RedBoot"
0x00040000-0x00100000 : "Linux kernel"
0x00040000-0x00080000 : "RedBoot[backup]"
```

```
0x00080000-0x00100000 : "unallocated space"
0x00100000-0x01f80000 : "JFFS2"
0x01f80000-0x01f81000 : "RedBoot config"
mtd: partition "RedBoot config" doesn't end on an erase block -- force read-only
0x01fc0000-0x02000000 : "FIS directory"
Linux Kernel Card Services 3.1.22
  options:  [pm]
SA-1100 PCMCIA (CS release 3.1.22)
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 1024 bind 1024)
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
NetWinder Floating Point Emulator V0.95 (c) 1998-1999 Rebel.com
FAT: bogus logical sector size 408
VFS: Mounted root (jffs2 filesystem).
Freeing init memory: 64K
INIT: version 2.78 booting
Mounting local filesystems...
```

Pour notre carte d'évaluation, nous utilisons le pilote MTD, décrit au chapitre précédent, associé au système de fichier journalisé JFFS2. La configuration du noyau (fichier **`.config`**) pour la partie MTD est donc la suivante :

```
#
# Memory Technology Devices (MTD)
#
CONFIG_MTD=y
CONFIG_MTD_PARTITIONS=y
CONFIG_MTD_REDBOOT_PARTS=y
CONFIG_MTD_CHAR=y
CONFIG_MTD_BLOCK=y

#
# RAM/ROM/Flash chip drivers
#
CONFIG_MTD_CFI=y
CONFIG_MTD_GEN_PROBE=y
CONFIG_MTD_CFI_ADV_OPTIONS=y
CONFIG_MTD_CFI_NOSWAP=y
CONFIG_MTD_CFI_GEOMETRY=y
CONFIG_MTD_CFI_B4=y
CONFIG_MTD_CFI_I2=y
CONFIG_MTD_CFI_INTELEXT=y

#
# Mapping drivers for chip access
#
CONFIG_MTD_SA1100=y
#
# File systems
```

```
#
CONFIG_JFFS2_FS=y
CONFIG_JFFS2_FS_DEBUG=0
```

Pour le démarrage, on obtient les traces suivantes :

```
<5>SA1100 flash: probing 32-bit flash bus
<7>0: offset=0x0,size=0x40000,blocks=128
<4>Using buffer write method
<5>Using RedBoot partition definition
<5>Creating 7 MTD partitions on "SA1100 flash":
<5>0x00000000-0x00040000 : "RedBoot"
<5>0x00040000-0x00100000 : "Linux kernel"
<5>0x00040000-0x00080000 : "RedBoot[backup]"
<5>0x00080000-0x00100000 : "unallocated space"
<5>0x00100000-0x01f80000 : "JFFS2"
<5>0x01f80000-0x01f81000 : "RedBoot config"
<4>mtd: partition "RedBoot config" doesn't end on an erase block -- force read-only
<5>0x01fc0000-0x02000000 : "FIS directory"
```

On note la détection des mémoires flash (2×128 Mbits, chacune sur 16 bits de largeur de données, l'association des deux formant donc un bus de 32 bits) par la couche MTD, puis la lecture des partitions de RedBoot stockées dans la partition *FIS directory*. Chacune des partitions est accessible à travers **`/dev/mtdblockX`**.

```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00040000 00040000 "RedBoot"
mtd1: 000c0000 00040000 "Linux kernel"
mtd2: 00040000 00040000 "RedBoot[backup]"
mtd3: 00080000 00040000 "unallocated space"
mtd4: 01e80000 00040000 "JFFS2"
mtd5: 00001000 00040000 "RedBoot config"
mtd6: 00040000 00040000 "FIS directory"
```

En résumé

L'utilisation d'un disque mémoire peut être conseillée dans le cas d'un système de petite taille qui ne doit pas sauvegarder de données durant son exécution. L'utilisation d'une image de root file system compressée est cependant assez contraignante durant la phase de développement de par les nombreux transferts qui sont effectués entre le poste de développement et la cible.

Linux fournit en standard des outils de mise au point de programme comme gdb, strace ou efence. Gdb permet aussi de mettre au point un programme à distance sur une cible connectée *via* réseau ou câble série.

LinuxBIOS est un projet Open Source destiné à remplacer le BIOS standard qui n'est pas utilisé par le noyau Linux. RedBoot est un outil puissant pour parer au développement et à la mise au point sur de nombreuses architectures.

Troisième partie

Mises en œuvre particulières

Cette partie décrit quelques techniques spécifiques de l'environnement embarqué sous Linux. Plusieurs chapitres seront consacrés à des versions spéciales de Linux comme les extensions temps-réel (RTLinux et RTAI) et la version pour micro-contrôleur µClinux. Un chapitre est consacré à la mise en œuvre d'interfaces graphiques adaptées. Pour finir, un chapitre sera dédié à l'utilisation d'environnements de développement croisé utilisables sur des plates-formes Linux et Windows et permettant de développer et de mettre au point du code sur des cibles embarquées.

9

Systèmes temps réel

Dans ce chapitre, on va s'attacher à décrire les principales solutions autorisant la mise en place d'une solution Linux pour un environnement temps réel. Le chapitre 1 a introduit de manière générale les concepts d'un système « temps réel » par opposition à un système dit à « temps partagé ». Nous rappellerons tout d'abord qu'il existe deux types de systèmes temps réel :

- Les systèmes dits temps réel « mous » ou « souples » (*soft real time*) pour lesquels les contraintes temps réel ne sont pas très fortes et surtout où le non-respect de l'échéance temporelle ne met pas en péril le système ni l'environnement. On parlera de temps de réponse d'« environ X unités de temps ». Ces systèmes peuvent se rapprocher d'un système classique comme Linux, ce dernier pouvant acquérir ce type de fonctionnalités après quelques modifications simples du noyau. Ils respectent en général des délais de réponse assez importants, de l'ordre de la milli-seconde.
- Les systèmes dits temps réel « durs » (*hard real time*) pour lesquels le respect de l'échéance temporelle est indispensable. Dans la plupart des cas, ces systèmes doivent également respecter les contraintes les plus fortes au niveau du délai de réponse (quelques dizaines de micro-secondes, parfois moins).

Tests sur un noyau Linux standard

Horloge temps réel `/dev/rtc`

Un noyau Linux standard dispose d'une horloge « temps réel » à travers le device **`/dev/rtc`** (*Real Time Clok*) associé à l'interruption de niveau 8. L'utilisateur peut se servir de ce device pour programmer une alarme ou une interruption périodique. Le fichier **`Documentation/rtc.txt`** des sources du noyau 2.4 donne une description précise des possibilités

de cette horloge. Le document contient également un petit programme de test **rtctest.c**. Après compilation, l'exécution du programme donne le résultat suivant :

```
# ./rtctest

                        RTC Driver Test Example.

Counting 5 update (1/sec) interrupts from reading /dev/rtc: 1 2 3 4 5
Again, from using select(2) on /dev/rtc: 1 2 3 4 5

Current RTC date/time is 25-5-2002, 15:04:28.
Alarm time now set to 15:04:33.
Waiting 5 seconds for alarm... okay. Alarm rang.

Periodic IRQ rate was 1024Hz.
Counting 20 interrupts at:
2Hz:     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
4Hz:     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
8Hz:     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
16Hz:    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
32Hz:    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
64Hz:    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

                         *** Test complete ***

Typing "cat /proc/interrupts" will show 131 more events on IRQ 8.
```

L'utilisation de **/dev/rtc** ne pose pas de problème lorsque le système est raisonnablement chargé. Quand le système est trop chargé, on constate de plus en plus de pertes d'interruption, et ce en fonction de la puissance du processeur et de la charge. Le document **rtc.txt** cite déjà des pertes d'interruption sur un 486-33 MHz au-delà d'une fréquence de 1024 Hz. Pour évaluer les performances d'un noyau Linux du point de vue du respect des contraintes temps réel, nous pouvons utiliser les outils latencytest et realfeel.

L'outil latencytest

L'outil latencytest (*http://www.gardena.net/benno/linux/audio*) est constitué d'une suite de programmes destinés à évaluer les problèmes de temps de latence sur un noyau Linux. La commande **latencytest** permet de jouer un échantillon sonore avec ou sans charge du système. Un autre programme **rtc_latencytest** permet d'évaluer le comportement de l'horloge temps réel précédemment décrite. L'outil permet de provoquer une charge du système au niveau affichage graphique (X11), accès à **/proc** et accès disque.

L'outil realfeel

Cet outil est inclus dans le paquetage amlat disponible sur *http://www.zip.com.au/~akpm/linux/schedlat.html*. La commande **realfeel** utilise l'horloge temps réel **/dev/rtc** afin de générer une sollicitation à 2048 Hz. La période donne le temps de réponse théorique du

système et chaque point de mesure représente le temps de latence entre la valeur théorique et la valeur réelle. Les résultats sont stockés dans un simple fichier d'historique qui permet de générer des graphiques en utilisant l'outil gnuplot (*http://www.gnuplot.info*).

Pour forcer la charge du système, on utilise les programmes `crashme` et `burnMMX` extraits de la suite Open Source CTCS (*Cerberus Test Control System*) disponible sur *http://sourceforge.net/projects/va-ctcs*. Une suite d'outils similaires existe également sous forme de paquetage RPM disponible sur *http://people.redhat.com/bmatthews/cerberus*.

Résultats du test

Les tests sont effectués sur un AMD Duron 700 MHz équipé de 360 Mo de mémoire vive. L'utilisation de `realfeel` sur cinq millions d'itérations (interruptions `/dev/rtc`) indique des temps de latence importants dans certains cas. La majorité des valeurs (plus de 99,97 %) se situe avant 5 ms mais quelques points se positionnent au-delà de 230 ms.

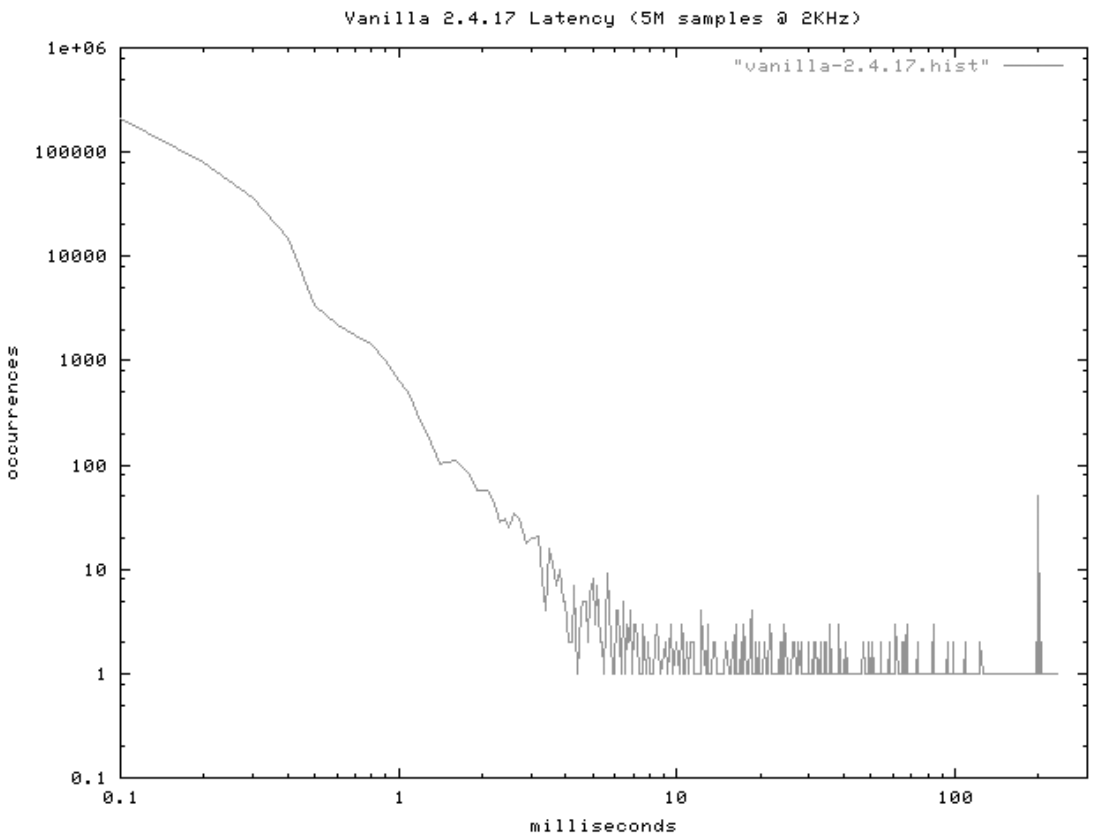


Figure 9-1

Mesure avec realfeel sur un noyau standard chargé.

```
maximum latency: 232.6ms
Mean: 0.0883230599960576ms
Standard Deviation: 2.11903578618566
92.84442% of samples < 0.1ms
97.08432% of samples < 0.2ms
99.73050% of samples < 0.5ms
99.84382% of samples < 0.7ms
99.94038% of samples < 1ms
99.97922% of samples < 5ms
99.98096% of samples < 10ms
99.98590% of samples < 50ms
99.98828% of samples < 100ms
100.00000% of samples < 232.7ms
```

Les différentes approches temps réel pour Linux

Le premier constat est sans appel. Linux dans sa version standard n'est *pas* un système temps réel mais un système à temps partagé. Même s'il fait une remarquable percée dans le monde des systèmes industriels et embarqués, il a été initialement conçu pour équiper des systèmes généralistes, en particulier des serveurs, pour lesquels la gestion du temps est totalement différente, le but d'un système à temps partagé étant de donner une impression de confort à ses utilisateurs. Tous les systèmes embarqués ne nécessitent pas de contraintes temps réel et nous avons vu que, quand Linux est correctement dimensionné dans un environnement d'applications connues, les temps de réponse sont à peu près stables. Si cette situation est satisfaisante pour l'application envisagée, nous pourrons facilement utiliser un noyau standard.

Si les contraintes de temps de réponse deviennent plus importantes, il faudra utiliser une version modifiée du noyau Linux. Les modifications peuvent être de deux types.

Modification de l'ordonnanceur

L'ordonnanceur Linux utilise trois types algorithmes possibles (*policies*). Ces algorithmes sont identifiés par les valeurs `SCHED_RR`, `SCHED_FIFO` et `SCHED_OTHER`. Le type `SCHED_OTHER` est utilisé pour les tâches non temps réel. Le type `SCHEDD_RR` utilise un algorithme de partage du temps associé à une valeur de priorité. Lorsque la tâche a terminé son quantum de temps, elle est placée de nouveau dans la file d'attente correspondant à sa priorité, et on lui alloue un nouveau quantum. Le type SCHED_FIFO n'utilise pas la notion de quantum de temps et la tâche est exécutée tant qu'elle n'est pas bloquée par une demande de ressource. Pour définir une tâche à la plus haute priorité possible, on peut utiliser le code suivant, qui est d'ailleurs repris par des outils de mesure comme latencytest et realfeel.

```
int set_realtime_priority(void)
{
  struct sched_param schp;
  /*
   * set the process to realtime privs
```

```
     */
    memset(&schp, 0, sizeof(schp));
    schp.sched_priority = sched_get_priority_max(SCHED_FIFO);

    if (sched_setscheduler(0, SCHED_FIFO, &schp) != 0) {
      perror("sched_setscheduler");
      exit(1);
    }

    return 0;
  }
```

Cela étant dit, l'ordonnanceur Linux standard n'a pas la granularité nécessaire à une gestion de tâches similaire à celle d'un noyau temps réel pour lequel les tâches sont gérées par des niveaux de priorité fixes.

La première approche consiste à modifier le système de gestion du temps à l'intérieur de noyau (ordonnanceur ou *scheduler*) de manière à améliorer le niveau de préemption et donc se rapprocher d'un comportement d'ordonnanceur temps réel. La solution utilisée par cette approche est d'ajouter des « points de préemption » de manière à vérifier le plus souvent possible si le passage à une tâche de plus haute priorité est nécessaire ; la fonction d'ordonnancement `schedule` est donc exécutée plus souvent. Cette solution est disponible sous forme du patch *kernel-preempt* pour le noyau 2.4 et elle est intégrée au noyau 2.6. La société MontaVista (*http://www.mvista.com*), spécialisée dans la diffusion de solutions Linux embarquées, est à l'origine de ce développement qui est disponible en Open Source.

Une approche similaire, présentant cependant quelques différences, est celle du patch *low-latency* qui, au lieu d'ajouter systématiquement des points de préemption, les place à des points stratégiques bloquant l'appel à l'ordonnanceur pendant une durée non négligeable. La difficulté dans cette approche reste d'identifier correctement les points en question. Toutefois, cette stratégie a l'avantage de limiter la chute de performance que pourraient provoquer un trop grand nombre d'appels à l'ordonnanceur.

Ces solutions ont surtout le gros avantage de ne pas modifier l'interface de programmation des applications car tout est fait au niveau du noyau et les développeurs travaillent toujours en mode utilisateur (et non en mode noyau), ce qui est une garantie de stabilité du système, une tâche utilisateur ne pouvant pas « planter » le noyau. En revanche, elles ont l'inconvénient de nécessiter des modifications dans un noyau qui n'est pas conçu au départ pour un ordonnancement temps réel des tâches.

Même si les résultats obtenus sont encourageants (voir à la fin du chapitre), il est judicieux de réserver cette technique à des systèmes à contraintes temps réel souples.

Ajout d'un véritable noyau temps réel

La seconde approche est celle défendue par les projets RTLinux (*http://www.fsmlabs.com*) et RTAI (*http://www.rtai.org*). Le principe simple qui y préside est de considérer que le

noyau Linux n'est pas conçu pour traiter des tâches temps réel et donc d'ajouter au système un noyau temps réel partageant la mémoire avec le noyau Linux. Ce noyau temps réel s'intercale entre le matériel et le noyau Linux de manière à récupérer les interruptions matérielles. Les fonctions comme *cli()* ou *sti()* sont redéfinies de manière à ce que toutes les interruptions soient préalablement traitées par la partie temps réel. Les pilotes standards du noyau Linux continuent à fonctionner mais ils doivent être recompilés s'ils utilisent des fonctions liées aux interruptions. Si les pilotes doivent assurer des fonctions temps réels, ils doivent également être adaptés pour RTLinux. La distribution RTLinux/GPL fournit en exemple la version temps réel *rt_com* d'un pilote de port série.

Le noyau temps réel est bien sûr préemptif et utilise un ordonnanceur qui contrairement au noyau Linux est basé sur la notion de priorité fixe. Les services habituels d'un système Linux sont toujours traités par le noyau standard mais les tâches temps réel (et seulement celles-ci) sont traitées par le noyau temps réel. Pour ce dernier, le noyau Linux est considéré comme une tâche de « plus faible priorité » (*idle task*). Le synoptique d'un tel système est présenté ci-après sous forme de schéma.

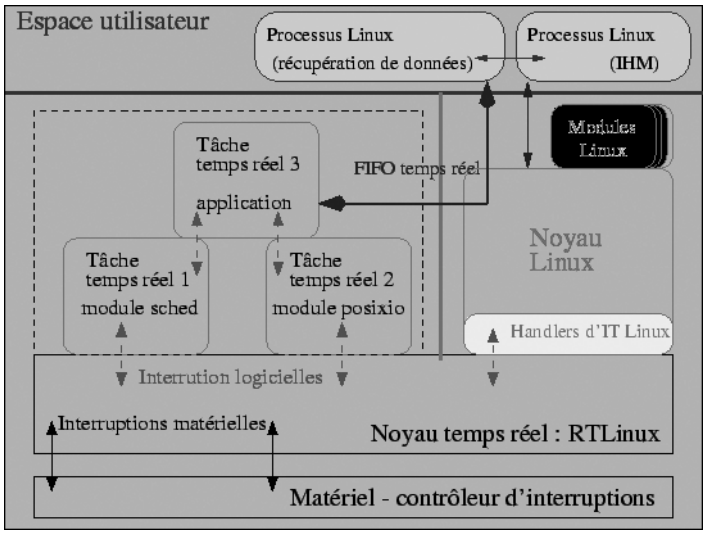


Figure 9-2
Architecture RTLinux.

Les tâches temps réel sont des modules au sens Linux du terme et donc exécutées dans le même espace que le noyau Linux (espace noyau et non espace utilisateur). Ce point est très intéressant au niveau des performances mais cela implique qu'une tâche temps réel mal programmée peut être fatale au système. Les tâches temps réel sont des threads ou « processus légers » au niveau noyau (*kernel threads*), la programmation de ces threads est conforme à la norme Posix.1c. En raison de l'utilisation de deux noyaux, le système

est très performant et le temps de réponse entre l'arrivée de l'interruption et le démarrage de la fonction associée est couramment inférieur à 15 micro-secondes.

D'une manière générale, la programmation dans l'espace noyau doit être *impérativement* limitée aux tâches temps réel. Tout service qui n'a pas de contrainte temps réel doit utiliser le noyau Linux standard.

Dans la suite du chapitre, on s'attachera à décrire quelques exemples illustrant les concepts exposés ici. Nous allons maintenant décrire en détail l'utilisation de RTLinux et RTAI, puis évoquer quelques patches préemptifs pour le noyau Linux.

L'outil de test rt_realfeel

L'outil rt_realfeel est une adaptation « maison » de l'outil realfeel cité précédemment. Il est utilisé pour évaluer les temps de latence dans le cas des systèmes Linux temps réel durs de type RTLinux ou RTAI. Le principe en est de générer une tâche périodique par la fonction `pthread_make_periodic_np`. La période donne le temps de réponse théorique du système et chaque point de mesure représente le temps de latence entre la valeur théorique et la valeur réelle. Un fichier historique de même type que celui de realfeel est généré à la fin de l'exécution.

Utilisation de RTLinux

RTLinux est un projet Open Source issu des travaux de Victor Yodaiken et Michael Barabanov à l'université du Nouveau-Mexique aux États-Unis. Les deux créateurs du projet ont depuis créé la société FSMLabs (*http://www.fsmlabs.com*) qui, en plus de la version GPL (RTLinux/GPL), fournit une version commerciale de RTLinux (RTLinux/Pro) ainsi que du support. Le projet est depuis toujours disponible sous GPL mais un brevet (*patent*) est lié au concept du double noyau de RTLinux. La licence d'utilisation de la version GPL est très claire à ce sujet. L'utilisation de RTLinux/GPL n'entraîne pas le paiement de droits d'utilisation à partir du moment où ce dernier est utilisé dans un projet lui-même diffusé sous GPL. Si le projet n'est pas diffusé sous GPL, il est nécessaire d'utiliser la RTLinux/Pro qui est également livrée avec les sources, qui ne sont cependant pas redistribuables comme c'est bien entendu le cas pour RTLinux/GPL. Les détails du brevet sont disponibles à l'adresse *http://www.fsmlabs.com/about/patent*.

RTLinux est clairement un produit orienté industrie et la documentation disponible sur le site de FSMLabs est très complète. Une FAQ traitant des domaines techniques et juridiques est disponible en ligne sur *http://www.fsmlabs.com/products/faq.htm* ou bien sous forme PDF sur la distribution de RTLinux/GPL. RTLinux est disponible pour les architectures x86, PowerPC, Alpha et MIPS.

Une version réduite de RTLinux (MiniRTL) est également disponible. Elle est basée sur un noyau Linux 2.2, RTLinux 2.0, et tient sur une seule disquette 1,44 Mo. MiniRTL est disponible en téléchargement sur *ftp://ftp.fsmlabs.com/pub/rtlinux/minirtl*.

Installation de RTLinux/GPL

La version 3.1 de RTLinux/GPL est disponible en téléchargement sur *ftp://ftp.fsmlabs.com/pub/rtlinux/v3*. Dans la suite du chapitre, nous avons utilisé l'archive `rtlinux-3.1.tar.gz`. La distribution se compose d'un patch à appliquer au noyau 2.4.4 ou au noyau 2.2.19 (versions supportées par RTLinux/GPL), des sources du noyau temps réel, de la documentation et de plusieurs exemples d'utilisation. Le fichier `Installation.txt` décrit, en anglais, la procédure d'installation.

L'installation s'effectue dans le répertoire `/usr/src` de manière à obtenir un répertoire `/usr/src/rtlinux-3.1` après extraction de l'archive. On peut également positionner un lien symbolique.

```
cd /usr/src
tar xzvf /tmp/rtlinux-3.1.tar.gz
ln -s rtlinux-3.1 rtlinux
```

Il est également nécessaire de récupérer le noyau Linux version 2.4.4 et d'extraire ce dernier dans le répertoire `/usr/src/rtlinux-3.1`. Nous rappelons que les sources des noyaux Linux officiels sont disponibles sur *ftp://ftp.kernel.org*. En France, le miroir *ftp://ftp.free.fr/mirrors/ftp.kernel.org/linux/kernel/v2.4* est très efficace. Lorsque l'archive `linux-2.4.4.tar.bz2` est disponible, on l'extrait dans le répertoire adéquat, ce qui a pour effet de créer le répertoire `linux`.

```
cd /usr /src/rtlinux-3.1
tar xjvf /tmp/linux-2.4.4.tar.bz2
```

Attention

RTLinux/GPL est prévu pour utiliser la version *officielle* du noyau Linux et non les versions modifiées distribuées par les éditeurs comme Red Hat.

On doit ensuite appliquer le patch RTLinux au noyau 2.4.4, soit :

```
cd /usr/src/rtlinux-3.1/linux
patch -p1 < ../kernel-patch-2.4.4
```

La prochaine étape est de configurer le noyau Linux comme décrit au chapitre 4 en utilisant `make xconfig` (ou bien *menuconfig* ou *config*). Pour cela, on fait :

```
cd /usr/src/rtlinux-3.1/linux
make xconfig
```

On doit alors valider les options nécessaires pour le système actuel (indépendamment de RTLinux). L'option `CONFIG_RTLINUX` ajoutée par le patch est validée automatiquement dans le cas d'une architecture x86 ou PowerPC.

Remarque

Il est *important* de ne *pas* valider le support APM car celui-ci peut interférer avec RTLinux.

On peut ensuite compiler le nouveau noyau en appliquant la procédure classique.

```
make dep
make clean
make bzImage
make modules
```

puis installer le noyau

```
cp arch/i386/boot/bzImage /boot/bzImage-2.4.4_rtl
make modules_install
```

et enfin ajouter l'entrée dans le fichier `/etc/lilo.conf`, puis valider par un appel à `/sbin/lilo`.

```
image=/boot/bzImage-2.4.4_rtl
  label=rtlinux
  read-only
  root=/dev/hda3
```

Il faut ensuite redémarrer le système en choisissant cette nouvelle image *rtlinux*. Lorsque le système a redémarré, il convient de vérifier que le fonctionnement classique est toujours correct. Si c'est le cas, on peut passer à la configuration puis à la génération du noyau temps réel. Pour ce faire, la procédure est similaire à celle utilisée pour le noyau Linux, soit :

```
make xconfig
make
make devices
make install
```

Cela a pour effet d'installer la distribution binaire sur `/usr/rtlinux` qui est un lien symbolique vers `/usr/rtlinux-3.1`.

La partie configuration utilise une interface graphique similaire à celle du noyau Linux. La partie Drivers permet de valider et configurer certaines fonctionnalités de communication décrites plus loin dans le chapitre.

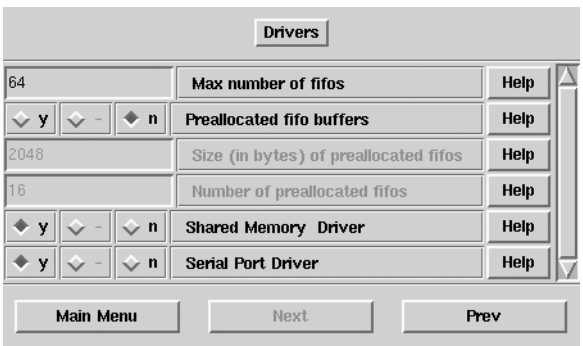


Figure 9-3

Configuration des drivers RTLinux.

Les exemples d'applications sont également installés dans ce répertoire. Le répertoire contient également le fichier **`rtl.mk`** qui pourra être inclus dans le fichier **`Makefile`** d'une application RTLinux afin d'ajouter les chemins d'accès aux fichiers d'en-tête C et options de compilation.

Pour terminer l'installation, il est conseillé de la tester avec le traditionnel programme *Hello World*. Avant d'exécuter un programme RTLinux, il est nécessaire de charger les modules correspondant au noyau temps réel, soit :

```
# /usr/src/rtlinux/bin/rtlinux start
Scheme: (-) not loaded, (+) loaded
  (+) mbuff
  (+) psc
  (+) rtl_fifo
  (+) rtl
  (+) rtl_posixio
  (+) rtl_sched
  (+) rtl_time
```

ou bien :

```
/usr/src/rtlinux/bin/rtlinux start nom_de_programme
```

À ce stade, on peut vérifier la présence des modules au moyen de **`lsmod`** ou **`rtlinux status`**.

```
# lsmod
Module                  Size  Used by
psc                    18816   0  (unused)
rtl_fifo                9952   0  [psc]
rtl_posixio             7168   0  [rtl_fifo]
rtl_sched              43104   0  [psc]
rtl_time                9984   0  [psc rtl_posixio rtl_sched]
rtl                    27200   0  [psc rtl_fifo rtl_posixio rtl_sched rtl_time]
mbuff                   6380   0  (unused)
nfs                    50792   1  (autoclean)
lockd                  39284   1  (autoclean) [nfs]
sunrpc                 66000   1  (autoclean) [nfs lockd]
eepro100               16656   1
rtc                     6552   0  (autoclean)

# rtlinux status

Scheme: (-) not loaded, (+) loaded
  (+) mbuff
  (+) psc
  (+) rtl_fifo
  (+) rtl
  (+) rtl_posixio
  (+) rtl_sched
  (+) rtl_time
```

Pour tester le programme **hello**, il suffit de faire :

```
cd /usr/src/rtlinux/examples/hello
make test
This is the simplest RTLinux program
First we remove any existing rtl-modules
You may see error warnings from "make" - ignore them
Type <return> to continue

rmmod sound
rmmod: module sound is not loaded
make: [test] Erreur 1 (ignorée)
rmmod rt_process
rmmod: module rt_process is not loaded
make: [test] Erreur 1 (ignorée)
rmmod frank_module
rmmod: module frank_module is not loaded
make: [test] Erreur 1 (ignorée)
(cd ../../; scripts/rmrtl)
Now insert the fifo and scheduler
Type <return> to continue

(cd ../../; scripts/insrtl)
Now start the real-time tasks  module
```

La trace de l'exécution est alors visible dans le fichier **/var/log/messages** ou bien avec la commande **dmesg**.

```
mbuff: kernel shared memory driver v0.7.2 for Linux 2.4.4-rtl
mbuff: (C) Tomasz Motylewski et al., GPL
mbuff: registered as MISC device minor 254
RTLinux Extensions Loaded (http://www.fsmlabs.com/)
I'm here; my arg is 0
I'm here; my arg is 0
I'm here; my arg is 0
```

On peut ensuite terminer l'application, ce qui correspond au déchargement du module associé.

```
Now let's stop the application
Type <return> to finish
```

Comme indiqué précédemment, l'exécution peut également être lancée, puis stoppée, plus simplement au moyen de la commande **rtlinux**.

```
rtlinux start hello
rtlinux stop hello
```

Introduction à l'API RTLinux

La documentation complète de l'API RTLinux est disponible en ligne au même endroit que la distribution RTLinux/GPL. Cette documentation est également accessible à l'adresse

http://www.fsmlabs.com/developers/man_pages. Un document *GettingStarted* donnant les informations indispensables au démarrage sous RTLinux est disponible à la même adresse.

Un exemple simple : Hello World

Comme cela a été déjà vu, la programmation de modules RTLinux s'effectue dans l'espace noyau en utilisant la technique des processus légers Posix. Le source **`hello.c`** du programme précédemment utilisé donne la structure minimale d'un programme RTLinux.

Tout comme un module chargeable pour le noyau Linux, un module RTLinux est constitué de deux entrées **`init_module`** et **`cleanup_module`**.

```
int init_module(void) {
  return pthread_create (&thread, NULL, start_routine, 0);
}

void cleanup_module(void) {
  pthread_delete_np (thread);
}
```

L'entrée **`init_module`** effectue le démarrage du thread principal associé à la fonction **`start_routine()`**. La seule manière d'arrêter une application RTLinux est de décharger le module associé. La fonction **`cleanup_module`** contient donc l'appel à la fonction de destruction du thread. On pourrait également faire :

```
void cleanup_module(void) {
   pthread_cancel (thread);
   pthread_join (thread, NULL);
}
```

Tout comme pour l'utilisation des threads Posix dans l'espace utilisateur, il est nécessaire d'effectuer un **`pthread_join`** sur le thread afin de libérer l'espace mémoire alloué.

Le code source de la fonction **`start_routine`** est donné ci-après.

```
void * start_routine(void *arg) {
   struct sched_param p;

   /* Init de la priorité à 1 */
   p.sched_priority = 1;
   pthread_setschedparam (pthread_self(), SCHED_FIFO, &p);

   /* Appel de l'ordonnanceur toutes les 500 µs */
   pthread_make_periodic_np (pthread_self(), gethrtime(),
                             500000000);

   /* Boucle infinie d'exécution de la tâche */
   while (1) {
     /* Attente de l'ordonnanceur */
     pthread_wait_np();
```

```
      /* Affichage du message */
      rtl_printf("I'm here; my arg is %x\n", (unsigned) arg);
    }
    return 0;
}
```

La première partie de la fonction fixe la valeur de la priorité à 1. L'appel à **pthread_make_periodic_np** indique à l'ordonnanceur d'exécuter le thread selon une période de 500 µs. La suite constitue le corps de l'exécution du thread, la fonction **pthread_wait_np** bloquant l'exécution du thread jusqu'à un nouvel appel de l'ordonnanceur (soit pendant 500 µs). L'appel à **rtl_printf** affiche simplement un message dans le système de trace du noyau, soit **/var/log/messages** ou **dmesg**.

Au niveau du fichier **Makefile**, il est nécessaire d'inclure le fichier **rtl.mk** généré lors de l'installation de RTLinux de manière à définir correctement les chemins d'accès des entêtes et définitions. Le **Makefile** pour l'application aura donc l'allure suivante :

```
all: hello.o

include ../../rtl.mk

clean:
  rm -f *.o
```

Communication inter-process (IPC)

Comme nous l'avons dit précédemment, la structure de RTLinux conduit à développer les applications en deux parties. La partie temps réel est définie dans l'espace du noyau mais doit être la plus courte et efficace possible. Elle correspond à un module du type précédemment décrit dans l'exemple *Hello World*. La partie noyau est souvent associée à une partie exécutée en mode utilisateur. Cette partie ne subit pas de contraintes temps réel car elle utilise le noyau Linux standard. Il existe divers moyens de communication entre la partie noyau et la partie utilisateur. L'ensemble de ces composants est appelé RTLinux IPC (*RTLinux Inter Process Communication*).

Le premier type de composant est la FIFO qui est similaire aux « tubes nommés » utilisés dans la programmation Linux classique. La FIFO est un canal de communication unidirectionnel et il est donc nécessaire d'utiliser deux FIFO pour établir une communication bidirectionnelle entre un module et une application utilisateur. Les FIFO sont associées à des devices en mode caractère, créés lors de l'installation de RTLinux par la commande **make devices**.

```
# ls -l /dev/rtf[0-5]
crw-r--r--    1 root     root     150,   0 mai 22 15:05 /dev/rtf0
crw-r--r--    1 root     root     150,   1 mai 22 15:05 /dev/rtf1
crw-r--r--    1 root     root     150,   2 mai 22 15:05 /dev/rtf2
crw-r--r--    1 root     root     150,   3 mai 24 17:28 /dev/rtf3
crw-r--r--    1 root     root     150,   4 mai 22 15:05 /dev/rtf4
crw-r--r--    1 root     root     150,   5 mai 22 15:05 /dev/rtf5
```

L'installation de RTLinux/GPL crée par défaut 64 FIFO (**rtf0** à **rtf63**). On peut cependant augmenter cette valeur dans la procédure de configuration **make xconfig**. Côté noyau, les FIFO sont créées en utilisant les fonctions suivantes de l'API RTLinux, lesquelles doivent être appelées dans la fonction **init_module**.

```
#include <rtl_fifo.h>
int rtf_create(unsigned int fifo, int size);
int rtf_destroy(unsigned int fifo);
```

On doit également associer une fonction de traitement (*handler*) à la réception de données sur la FIFO. Pour ce faire, on fait appel à la fonction suivante :

```
#include <rtl_fifo.h>
int rtf_create_handler(unsigned int fifo, int (* handler)());
```

Côté utilisateur, la FIFO est utilisée comme un fichier classique :

```
int fd_fifo
fd_fifo = open ("/dev/rtf0", O_WRONLY);
```

Lorsque le descripteur de fichier est créé, on peut bien sûr effectuer les actions habituelles comme **read**, **write** ou bien **select**. Le petit exemple suivant est une version modifiée du programme de test **hello**. Le thread noyau se met en attente de caractère sur la FIFO **/dev/rtf0** ; à réception d'un caractère, le thread est réveillé par l'ordonnanceur et affiche un message.

Les fonctions **init_module** et **cleanup_module** ont maintenant l'allure suivante :

```
int init_module(void) {
  int f;
  rtf_destroy (0);
  f = rtf_create (0, 100);
  rtf_create_handler(0, &my_fifo_handler);
  return pthread_create (&thread, NULL, start_routine, 0);
}

void cleanup_module(void) {
  rtf_destroy(0);
  pthread_delete_np (thread);
}
```

Le *handler* de la FIFO est appelé sur réception d'un caractère. Il se résume simplement à la lecture du caractère (grâce à la fonction **rtf_get**) puis au réveil du thread principal.

```
int my_fifo_handler (unsigned int fifo)
{
  int err;

  while ((err = rtf_get (0, &c, 1)) == 1) {
    rtl_printf("my_fifo_handler: got %x %c\n", c, c);
    pthread_wakeup_np (thread);
  }
```

```
  if (err != 0)
     return -EINVAL;
  else
    return 0;
}
```

La fonction **start_routine** se contente d'afficher un message au réveil.

```
void * start_routine(void *arg)
{
  struct sched_param p;
  p.sched_priority = 1;
  pthread_setschedparam (pthread_self(), SCHED_FIFO, &p);

  while (1) {
    pthread_wait_np ();
    rtl_printf("start_routine: Got char %x\n", c);
  }
  return 0;
}
```

Un exemple un peu plus complet d'utilisation des FIFO est disponible dans les sources de RTLinux dans le répertoire **examples/frank**.

Le deuxième type de composant est la mémoire partagée, ou *shared memory*. L'utilisation de mémoire partagée est intéressante lorsque le débit des données est supérieur aux possibilités des FIFO explicitées précédemment. Des mesures concernant ce débit donnent environ 44 Mb/s pour un Pentium, 200 MHz et 137 Mb/s pour un AMD Athlon 800 MHz. Ces débits sont intéressants mais certaines cartes de mesure peuvent générer des débits de l'ordre de 40 Mb/s ou plus, ce qui se situe à la limite des possibilités des FIFO. Le concept de mémoire partagée entre deux processus utilisateur existe sous Linux depuis longtemps (voir **man shmop**) ; il est utilisé pour la communication entre le serveur X Window et les clients dans le cas d'une station de travail. Le partage de mémoire sous RTLinux s'effectue entre un module noyau et une application utilisateur.

L'accès à la mémoire partagée était initialement assez complexe ; un document en explique la procédure à l'adresse *http://www.isd.cme.nist.gov/projects/emc/shmem.html*. Pour simplifier les choses, il est plutôt recommandé d'utiliser le pilote *mbuff* livré avec la version 3 de RTLinux/GPL. Les sources du pilote sont disponibles sur **/usr/src/rtlinux/drivers/mbuff**. Afin de pouvoir utiliser le pilote, le support doit être validé lors de la configuration de RTLinux par **make xconfig**. Le pilote est associé à la présence du device **/dev/mbuff**.

```
# ls -l /dev/mbuff
crw-r--r--    1 root     root      10, 254 mai 22 15:05 /dev/mbuff
```

L'utilisation de **mbuff** est très simple et basée sur deux fonctions principales d'allocation et de libération.

```
#include <mbuff.h>
void * mbuff_alloc(const char *name, int size);
void mbuff_free(const char *name, void * mbuf);
```

Côté module noyau, on doit allouer le segment de mémoire partagée dans la fonction **`init_module`**. Le segment sera libéré dans **`cleanup_module`**.

```
volatile int* shm_ptr;
#define SHRMEM_NAME "mbufftest"

int init_module(void){

  if((shm_ptr = (volatile int*) mbuff_alloc(SHRMEM_NAME,sizeof(int))) == NULL)
  {
    rtl_printf("MBUFF_ALLOC FAILED\n");
    return -1;
  }

  return pthread_create (&thread,NULL,routine,0);
}

void cleanup_module(void)
{
  pthread_delete_np(thread);
  mbuff_free(SHRMEM_NAME,(void*)shm_ptr);
}
```

Dans notre exemple, le thread principal effectue simplement une incrémentation du contenu de la mémoire pointée par **`shm_ptr`** au fur et à mesure de l'exécution.

```
void* routine(void* t)
{
  struct sched_param p;
  p.sched_priority = 1;
  pthread_setschedparam (pthread_self(), SCHED_FIFO, &p);
  *shm_ptr = 0;

  pthread_make_periodic_np(pthread_self(),gethrtime(), 400000000);
  while(1)
  {
    pthread_wait_np();
    (*shm_ptr)++;
  }
  return 0;
}
```

Côté utilisateur, l'application se contente d'afficher le contenu du segment partagé. Il est intéressant de noter que les fonctions de manipulation de **`mbuff`** sont les mêmes côté noyau et côté utilisateur.

```
#include <stdio.h>
#include <errno.h>
#include <mbuff.h>

#define SHRMEM_NAME "mbufftest"
```

```
int main(void)
{
  int i;
  volatile int *shm_ptr;
  printf("NOW MBUFF ALLOCATE...\n");

  if( (shm_ptr = (volatile int*) mbuff_alloc(SHRMEM_NAME,sizeof(int)) ) == NULL)
  {
    fprintf(stderr,"MBUFF_ALLOC FAILED\n");
    exit(-1);
  }
    printf("ALLOC OK\n");

  for(i = 0;i < 10;i++)
  {
    sleep(1);
    printf("SHR MEM IS %d\n",*shm_ptr);
  }

  mbuff_free(SHRMEM_NAME,(void*)shm_ptr);
  return 0;
}
```

Mesure des temps de latence

Nous avons effectué un test my_amlat à 2048 Hz pendant 5 minutes sur un système non chargé, puis chargé. Nous remarquons que les temps ne dépassent pas les 10 µs (1 seul point à l'extérieur), et ce presque indépendamment de la charge. Le comportement de RTLinux en mode noyau est donc peu influencé par la charge du système (voir figure 9-4, page suivante).

Utilisation de RTAI

RTAI (*http://www.rtai.org*) est un projet GPL issu d'une branche de RTLinux. Le projet est maintenu par l'école polytechnique de Milan (*Politecnico di Milano*) en Italie. Les fonctionnalités et l'API de RTAI ont de fortes similitudes mais on ne peut pas affirmer qu'il y ait une compatibilité totale de RTAI avec l'API de RTLinux. Le fichier en-tête de compatibilité `rt_compat.h` (livré avec les sources de RTAI) permet cependant de parvenir à une compatibilité partielle pour des applications simples. RTAI partage avec RTLinux la compatibilité Posix, l'utilisation de FIFO de communication ainsi que la mémoire partagée. RTAI inclut en plus le concept LXRT (*LinuX Real Time*) qui permet d'utiliser des applications temps réel dans l'espace utilisateur, ce qui n'est pas le cas de RTLinux.

Contrairement à RTLinux, il n'existe pas de version commerciale de RTAI et de ce fait son approche est moins « industrielle » que celle de RTLinux. Le projet lui-même inclut beaucoup plus de composants que RTLinux, dans l'esprit plus *bazaar* typique de certains projets Open Source.

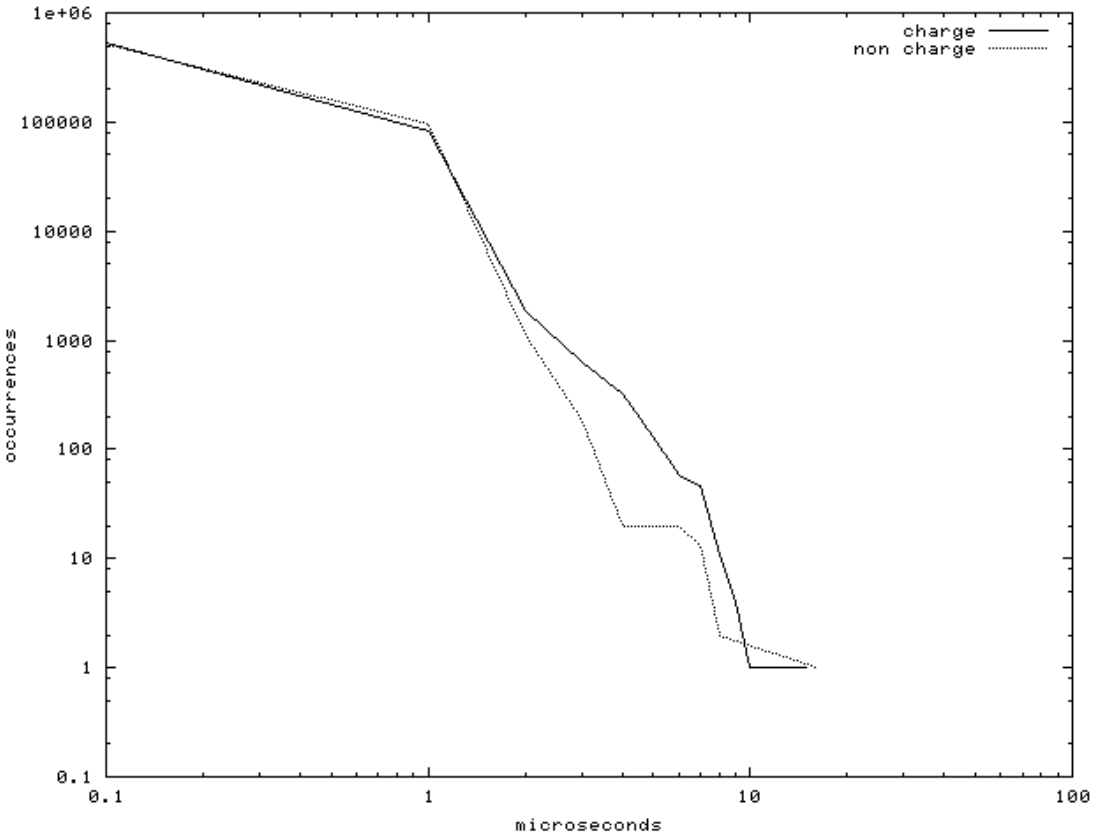


Figure 9-4
Test de latence RTLinux.

Les dernières versions de RTAI (3.1 et Fusion) ont cependant conduit à assainir la structure de la distribution. De même, ces dernières versions ne sont plus basées sur la RTHAL (pour *Real Time Hardware Abstraction Layer*) qui posait un problème vis-à-vis du brevet logiciel déposé par les créateurs de RTLinux. RTAI utilise aujourd'hui une couche basse nommée ADEOS (pour *Adaptative Domain Environment for Operating Systems*). Des informations sur ADEOS sont disponibles à l'adresse *http://home.gna.org/adeos*.

RTAI est désormais libéré de cette menace de brevet. De plus, sachant que RTLinux est devenu un produit quasiment propriétaire (la version GPL n'évoluant plus), RTAI est aujourd'hui le choix de prédilection pour les utilisateurs de temps réel dur sous Linux.

Dans la suite du chapitre nous détaillerons l'installation de RTAI-24.1.8 mais également l'installation de RTAI-3.1 qui constitue actuellement la dernière version stable qui plus est utilisable sur les noyaux 2.4 et 2.6.

Installation de RTAI-24.1.x

La distribution RTAI peut être téléchargée depuis le site *http://www.aero.polimi.it/RTAI*. Nous avons utilisé la version 24.1.8. Après extraction de l'archive `rtai-24.1.8.tgz`, on obtient le répertoire `/usr/src/rtai-24.1.8`.

Premier point important, le fichier `GCC-WARNING` indique que RTAI ne peut pas utiliser la version gcc-2.96 ou 3.x, ce qui pose un problème aux utilisateurs de distributions Red Hat récentes (7.2 en particulier) car c'est le compilateur installé par défaut. La version de compilateur conseillée est la 2.95.3 qui est plus ancienne. Après diverses pérégrinations, il apparaît que le plus simple (sur RH 7.2) est de télécharger les sources du compilateur requis sur le site *ftp://ftp/gnu/org/gnu/gcc* et d'installer ce dernier sur un répertoire différent de la version installée par le système comme `/usr/local`. L'accès au nouveau compilateur se fera simplement en plaçant `/usr/local/bin` *avant* `/usr/bin` dans la variable `PATH` locale.

```
# gcc --version
2.96
# export PATH=/usr/local/bin:$PATH
# gcc --version
2.95.3
```

La compilation de gcc est relativement simple. Après extraction de l'archive, il suffit de se positionner dans le répertoire des sources et d'exécuter :

```
./configure
make
make install
```

Tout comme RTLinux, RTAI est constitué de patches du noyau Linux associés à un noyau temps réel sous forme de modules. La version actuelle de RTAI est prévue pour fonctionner avec les noyaux 2.2.17, 2.4.16 ou bien 2.4.17. Après extraction du noyau choisi (2.4.17) et installation sur `/usr/src/linux-2.4.17_rtai`, on peut appliquer le patch RTAI en exécutant :

```
cd /usr/src/linux-2.4.17_rtai
patch -p1 < /usr/src/rtai-24.1.8/patches/patch-2.4.17-rthal5g
```

Attention

Tout comme RTLinux, RTAI est prévu pour utiliser la version *officielle* du noyau Linux et non les versions modifiées distribuées par les éditeurs.

On doit ensuite configurer le noyau Linux avec `make xconfig` en prenant soin de valider l'option `CONFIG_RTHAL` dans le menu *Processor type and features*.

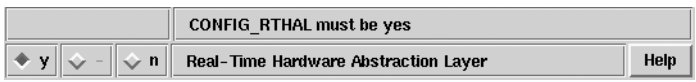
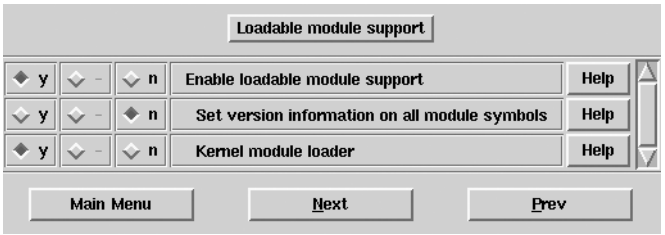


Figure 9-5

Validation du support RTAI.

De même, il ne faut *pas* valider l'option *Set version information on all module symbols* dans le menu *Loadable module support*.

Figure 9-6
Configuration du support des modules.



La compilation se fait ensuite par l'enchaînement classique :

```
make dep
make clean
make bzImage
make modules
```

et l'installation par :

```
make modules_install
cp arch/i386/boot/bzImage /boot/bzImage-2.4.17_rtai
```

De la même manière que pour RTLinux, on doit définir une nouvelle entrée dans le fichier **`/etc/lilo.conf`**, puis valider par un appel à **`/sbin/lilo`**.

```
image=/boot/bzImage-2.4.17_rtai
  label=rtai
  read-only
  root=/dev/hda3
```

Après redémarrage du système sur ce nouveau noyau, on doit maintenant configurer la partie noyau temps réel. Pour cela, on fait :

```
cd /usr/src/rtai-24.1.8
make menuconfig
```

pour configurer RTAI, ou bien :

```
make oldconfig
```

si l'on désire conserver la configuration par défaut. On peut aussi lancer le script **`configure`** qui lance ensuite la commande **`make config`**.

On effectue ensuite la compilation et l'installation en exécutant :

```
make modules
make
make install
make dev
```

On peut maintenant effectuer le test que propose un exemple livré avec la distribution :

```
# cd examples
# ./run
```

ce qui lance l'application qui s'affiche dans **`/var/log/messages`** et les traces du noyau.

```
# dmesg
***** RTAI NEWLY MOUNTED (MOUNT COUNT 1) ******

<>>> FP RESULT CHECK 35648996 <<<>
<>RT_HAL time: 6 s, AvrJ: 1, MaxJ: 2 us (35648996,-1043,0)<> 3001 0
<>RT_HAL time: 9 s, AvrJ: 0, MaxJ: 2 us (35648996,-1043,0)<> 6002 0
<>RT_HAL time: 12 s, AvrJ: 1, MaxJ: 2 us (35648996,-1043,0)<> 9002 0
<>RT_HAL time: 15 s, AvrJ: 0, MaxJ: 2 us (35648996,-1043,0)<> 12003 0
```

On peut stopper l'application avec **`./rem`**.

```
# ./rem
already root
+sh -c "rmmod -r ex_timer"
# dmesg

***** RTAI UNMOUNTED (MOUNT COUNT 1) ******
```

Mesure des temps de latence

Nous effectuons le même test qu'avec RTLinux (système non chargé puis chargé).

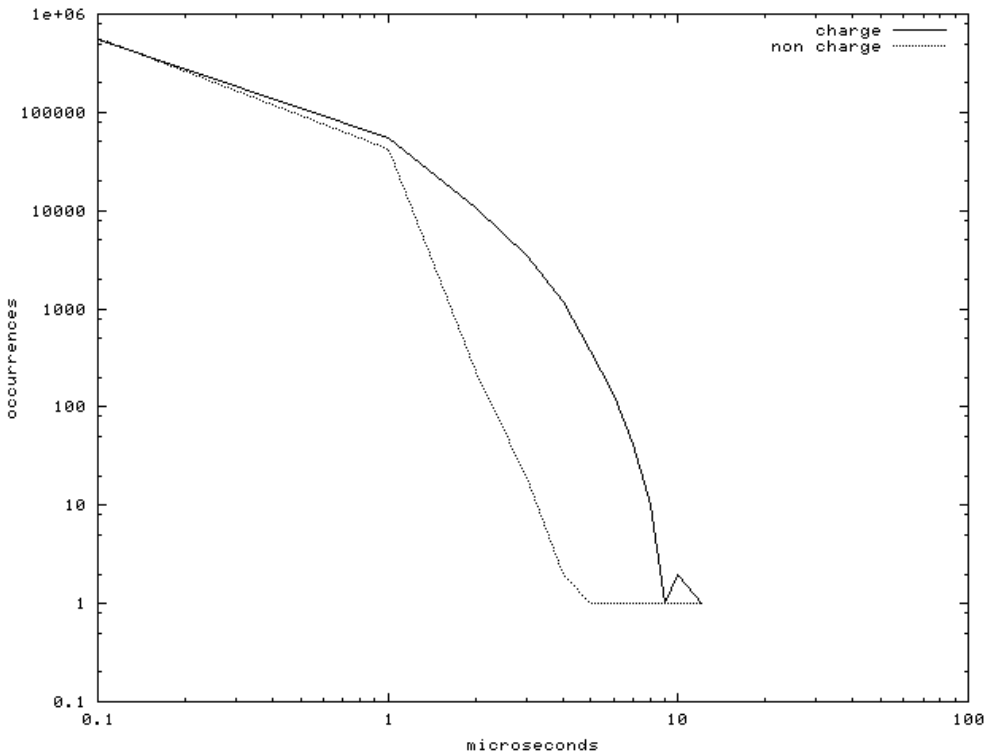


Figure 9-7

Test de latence RTAI.

Installation de RTAI-3.1

On peut télécharger la distribution à partir de l'adresse *http://www.rtai.org* en suivant le lien *Download* puis *Stable*. Contrairement à la version précédente, RTAI-3.1 supporte le noyau 2.6 et il est possible de compiler la distribution avec les dernières versions de *gcc* (3.x), ce qui n'était pas possible avec les versions 24.1.x.

Le fichier README.INSTALL donne les indications nécessaires à l'installation de la distribution soit :

1. Sélectionner un noyau Linux compatible avec RTAI et appliquer le patch correspondant. La liste des patches est disponible sur le répertoire `rtai-core/arch/i386/patches`.
2. Valider le support ADEOS dans le noyau via le menu *General setup/Adeos support*. En général, le support ADEOS sera configuré en module.
3. Compiler ce noyau, l'installer et vérifier le bon fonctionnement du système.
4. Exécuter le script `configure` du répertoire des sources RTAI-3.1. À la fin de son exécution, le script doit démarrer une interface graphique permettant de sélectionner les options de compilation. Dans un premier temps, le mieux est d'utiliser les options par défaut.
5. Compiler puis installer la distribution par `make` puis `make install`. La distribution sera installée par défaut sur `/usr/realtime`.

Lorsque la distribution est correctement installée, on peut tester les programmes de démonstration disponibles dans le répertoire `/usr/realtime/testsuite`. Le répertoire `kern` correspond aux exemples exécutés dans l'espace noyau, le répertoire `user` correspond aux exemples exécutés dans l'espace utilisateur (LXRT).

On peut effectuer un test de latence en mode noyau par la séquence suivante. Nous chargeons le système en exécutant en même temps un petit script d'écriture de fichier.

```
# cd /usr/realtime/testsuite/kern/latency
# ./run
*
*
* Type ^C to stop this application.
*
*

## RTAI latency calibration tool ##
# period = 100000 (ns)
# avrgtime = 1 (s)
# check overall worst case
# do not use the FPU
# start the timer
# timer_mode is oneshot


2005/04/10 19:39:58   min:    -2171    max:    15201    average:     1661
```

```
2005/04/10 19:39:59   min:    -3609    max:    15278    average:      -29
2005/04/10 19:40:00   min:    -3995    max:    15278    average:      123
2005/04/10 19:40:01   min:    -3995    max:    15543    average:       53
2005/04/10 19:40:02   min:    -3995    max:    15543    average:       99
2005/04/10 19:40:03   min:    -3995    max:    20759    average:       72
2005/04/10 19:40:04   min:    -3995    max:    20759    average:     1604
2005/04/10 19:40:05   min:    -3995    max:    20759    average:        5
2005/04/10 19:40:06   min:    -3995    max:    20759    average:       70
2005/04/10 19:40:07   min:    -3995    max:    20759    average:       53
2005/04/10 19:40:08   min:    -3995    max:    20759    average:       17
2005/04/10 19:40:08   min:    -3995    max:    20759    average:       17
```

Nous constatons que la latence maximale est d'environ 20 µs.

Pour le programme de mesure de latence en mode utilisateur, nous obtenons le résultat suivant :

```
# cd /usr/realtime/testsuite/kern/latency
# ./run
*
*
* Type ^C to stop this application.
*
*
## RTAI latency calibration tool ##
# period = 100000 (ns)
# average time = 1 (s)
# check overall worst case
# use the FPU
# start the timer
# timer_mode is oneshot

*** min: -6063, max: 17946 average: 1123  <Hit [RETURN] to stop> ***
*** min: -14487, max: 17946 average: 3412  <Hit [RETURN] to stop> ***
*** min: -14487, max: 21381 average: 1456  <Hit [RETURN] to stop> ***
*** min: -14487, max: 21381 average: 1127  <Hit [RETURN] to stop> ***
*** min: -14487, max: 21381 average: 1108  <Hit [RETURN] to stop> ***
*** min: -14487, max: 21381 average: 1098  <Hit [RETURN] to stop> ***
*** min: -14487, max: 21381 average: 1053  <Hit [RETURN] to stop> ***
*** min: -14487, max: 31546 average: 3985  <Hit [RETURN] to stop> ***
*** min: -14487, max: 31546 average: 1017  <Hit [RETURN] to stop> ***
*** min: -14487, max: 31546 average: 1142  <Hit [RETURN] to stop> ***
```

La latence est plus importante (environ 30 µs soit 10 µs de plus) mais le programme de test est exécuté dans l'espace utilisateur, ce qui a d'énormes avantages.

- Le développement et la mise au point des programmes en mode utilisateur sont beaucoup plus simples.
- Dans l'espace utilisateur, les problèmes liés à la GPL sont moins fréquents.

Remarque

Pour le dernier point concernant la GPL, rappelons que Linus Torvalds, créateur du noyau Linux considère que tout fichier contenant une inclusion (un #include en C) d'un fichier du noyau doit être diffusé sous GPL. On peut lire sur *http://seclists.org/lists/linux-kernel/2003/Dec/1042.html* la phrase

BUT YOU CAN NOT USE THE KERNEL HEADER FILES TO CREATE NON-GPL'D BINARIES

Il est donc préférable de programmer les tâches temps réel dans l'espace utilisateur !

Utilisation des patches du noyau

Les patches du noyau constituent une solution intermédiaire entre un noyau standard utilisant un ordonnanceur à temps partagé et un véritable noyau temps réel. Ces patches sont légers et dépendent de la version du noyau utilisé. On peut les appliquer simplement en utilisant une ligne de commande du type :

```
cd /usr/src/linux-2.4.mon_noyau
patch -p1 < nom_du_fichier_patch
```

Ils sont associés à la validation d'une option de préemption au niveau de la configuration du noyau intervenant avec **`make xconfig`**. L'utilisation d'un noyau modifié de la sorte ne transforme pas Linux en système temps réel mais permet d'améliorer les temps de latence dans les limites d'une utilisation acceptable pour une application donnée, et ce en conservant une interface de programmation classique en mode utilisateur.

Le patch preempt-kernel-rml

Ce patch est maintenu par Robert M. Love (*rml@tech9.net*) en collaboration avec la société MontaVista (*http://www.mvista.com*). Les patches, ainsi que différents outils, fichiers de documentation et résultats, sont disponibles sur *http://www.tech9.net/rml/linux*.

Pour le noyau 2.4, il faut valider l'option *Preemptible Kernel* du menu *Processor type and features* de la configuration du noyau. Une fonctionnalité similaire est désormais intégrée au noyau 2.6 au niveau du menu *Processor type and features/Preemptible kernel.*

Dans le cas de l'utilisation de la commande **`realfeel`**, on obtient le graphique suivant, qui est à comparer avec la figure 9-1 correspondant à un noyau standard. Une très grande majorité des valeurs reste inférieure à 1 ms avec un maximum à 45 ms au lieu des plus de 230 ms précédemment constatés.

```
maximum latency: 45.2ms
Mean: 0.0528876799956937ms
Standard Deviation: 0.0461523751362407
97.95326% of samples < 0.1ms
99.55722% of samples < 0.2ms
99.97026% of samples < 0.5ms
99.98960% of samples < 0.7ms
99.99650% of samples < 1ms
99.99954% of samples < 5ms
99.99982% of samples < 10ms
100.00000% of samples < 45.3ms
```

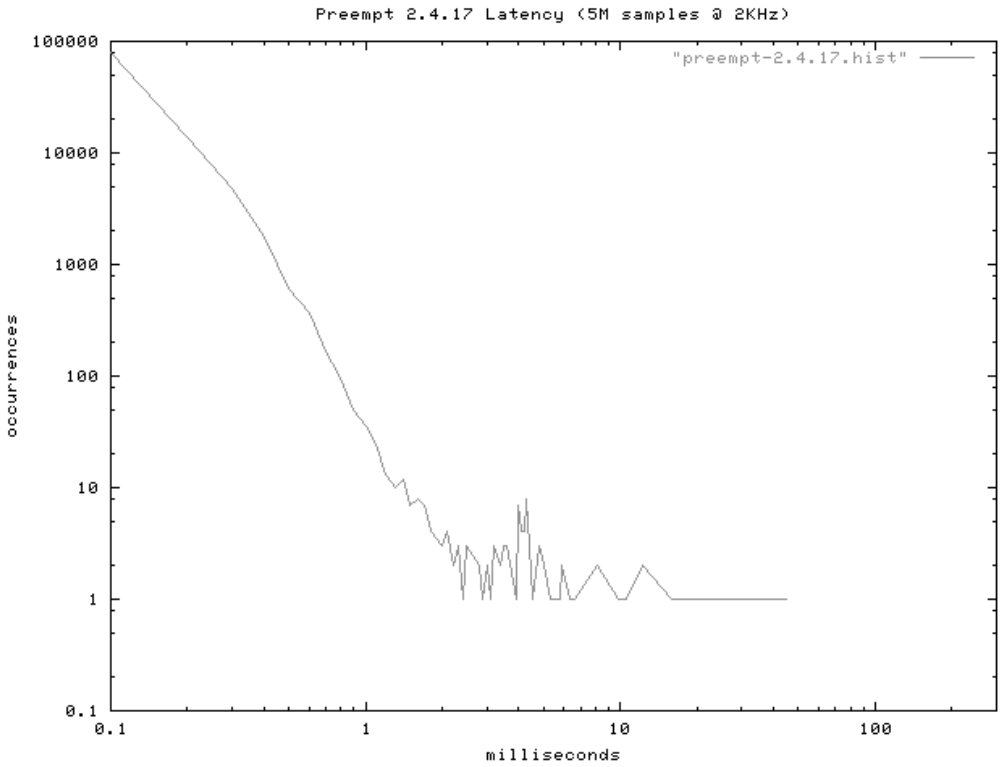


Figure 9-8

Test realfeel sur un noyau avec patch preempt-kernel.

Le patch low-latency

Initialement développé par Ingo Molnar, ce patch est dorénavant maintenu par Andrew Morton (*akpm@zip.com.au*). Il est disponible pour le noyau 2.4 sur le même site que realfeel, soit *http://www.zip.com.au/~akpm/linux/schedlat.html*.

Une fonctionnamlité similaire est désormais intégrée au noyau 2.6 au niveau du menu *Processor type and features/Preemptive kernel*.

Après application du patch, il est nécessaire de valider le support *Low latency scheduling* dans le menu *Processor type and features*. L'option *Control latency with sysctl* permet de valider ou non la fonction de manière dynamique en utilisant l'entrée **`/proc/sys/kernel/low_latency`**.

Testé avec le programme **`realfeel`**, le noyau 2.4.17 utilisé donne de très bons résultats sur cinq millions d'interruptions, la valeur maximale ne dépassant pas 1,4 ms.

```
maximum latency: 1.3ms
Mean: 0.0542797399957571ms
Standard Deviation: 0.025220506601371
```

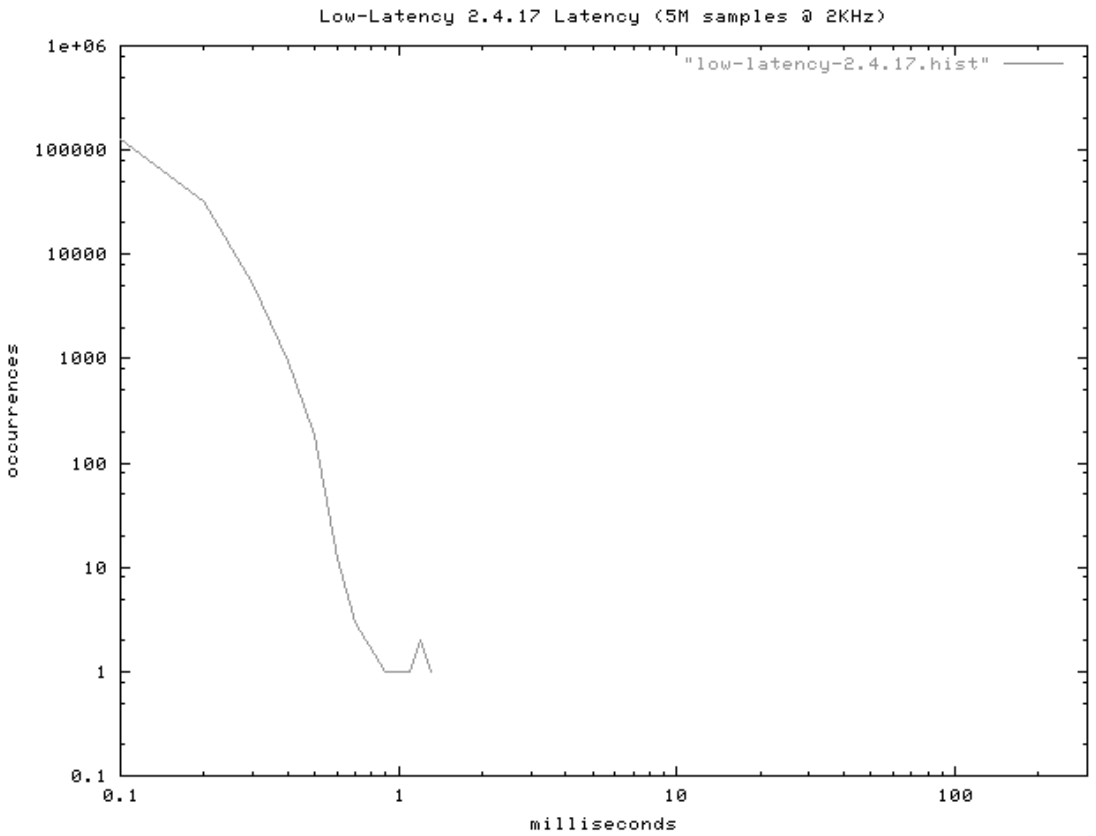


Figure 9-9

Test realfeel sur noyau avec patch low-latency.

```
96.66250% of samples < 0.1ms
99.21158% of samples < 0.2ms
99.99592% of samples < 0.5ms
99.99984% of samples < 0.7ms
99.99992% of samples < 1ms
100.00000% of samples < 1.4m
```

Combinaison de patch

Il est possible de combiner les patches preempt-kernel et low-latency car ils ne touchent pas les même portions de code dans le noyau Linux. Le fait de combiner l'approche du premier, qui tend à ajouter des points de préemptions systématiques, avec celle du second, qui tend à réduire les longs intervalles de temps sans appel à l'ordonnanceur, semble donner de bons résultats. Après application du patch preempt-kernel puis low-latency, on obtient le graphe suivant sur cinq millions d'interruptions. On note surtout que la combinaison des deux patches permet d'augmenter le nombre de valeurs inférieures à 100 µs.

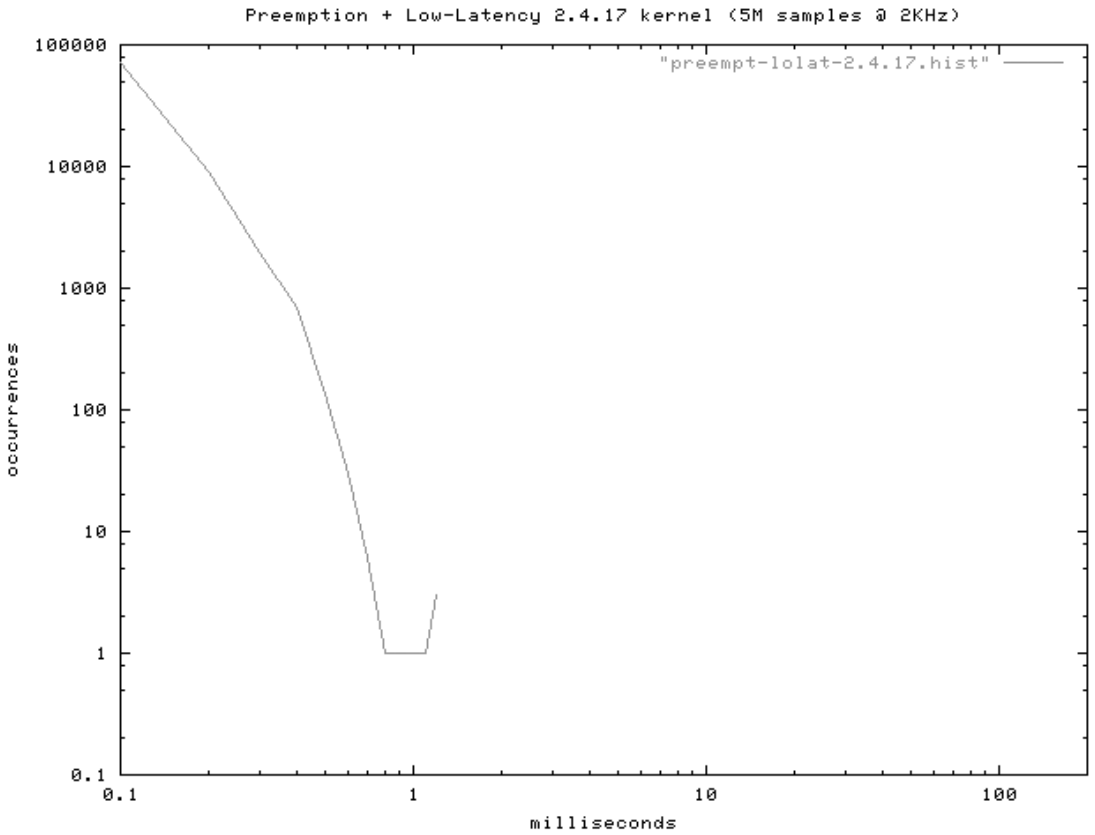


Figure 9-10

Test realfeel sur la combinaison des deux patches.

```
minimum latency: < 0.1ms
maximum latency: 1.2ms
Mean: 0.0520061799956548ms
Median: 0.05ms
Mode: 0.05ms (occurred 4915634 times)
Variance: 0.000282321298762259
Standard Deviation: 0.0168024194318038
98.31268% of samples < 0.1ms
99.76028% of samples < 0.2ms
99.99648% of samples < 0.5ms
99.99978% of samples < 0.7ms
99.99992% of samples < 1ms
100.00000% of samples < 1.3ms
```

Au moment de la rédaction de ces lignes, il semblerait qu'un patch unique combinant les deux fonctionnalités soit envisagé.

Le patch rtsched

Le projet rtsched (*http://rtsched.sourceforge.net*) vise (visait) à modifier l'ordonnanceur Linux de manière à séparer les tâches temps réel des autres tâches. De ce fait, le type `SCHED_OTHER` n'est pas traité de la même manière que les types `SCHED_RR` et `SCHED_FIFO`. Le traitement des quantums de temps est également optimisé. La validation s'effectue au niveau de la configuration du noyau dans le menu *Processor type and features*, ou il faut valider l'option *Real Time Scheduler* et définir la valeur maximale de priorité correspondant à une tâche temps réel. Les tâches de priorité 1 jusqu'à cette valeur seront considérées par le système comme des tâches temps réel. Le patch est disponible jusqu'à la version 2.4.16 du noyau Linux et il semble que le projet (également sponsorisé par MontaVista) n'évolue plus et que les approches précédentes aient pris le pas sur rtsched.

En résumé

Le noyau Linux standard utilise un ordonnancement à temps partagé et une forte charge du système peut induire des temps de latence importants et surtout imprévisibles.

Le comportement peut être amélioré par des patches comme kernel-preempt ou low-latency qui réduisent ces temps de latence mais ne transforment pas Linux en vrai système temps réel. Il est donc prudent de réserver cette technique à des applications temps réel souples. L'avantage principal en est l'API de programmation qui n'est pas modifiée par rapport à un noyau Linux standard.

Les produits RTLinux et RTAI utilisent une technique d'ajout d'un véritable noyau temps réel en parallèle au noyau Linux, ce dernier étant considéré comme une tâche de plus faible priorité du noyau temps réel. Cette technologie permet de mettre en place des systèmes temps réel à contraintes dures. Les tâches temps réel doivent cependant être développées avec une API spéciale dans l'espace noyau et non dans l'espace utilisateur.

Une version GPL de RTLinux peut être utilisée, mais seulement pour des applications GPL. La nouvelle version GPL de RTLinux n'est plus très à jour par rapport à la version propriétaire. Il est donc préférable d'utiliser RTAI qui est un véritable projet libre, offrant des performances similaires. Il permet de programmer des tâches temps réel dans l'espace utilisateur, ce qui évite les problèmes de conformité avec la GPL.

10

Systèmes minimaux : µClinux

Présentation de µClinux

Le projet µClinux (à prononcer « *you see LINUX* ») est un portage du noyau Linux pour les processeurs dépourvus de MMU (*Memory Management Unit*, voir chapitre 3). Le projet a démarré en 1998 par le portage d'un noyau 2.0 sur un processeur Motorola M68328 mais le produit existe maintenant en version 2.4 et 2.6.

La page d'accueil du projet est située à l'adresse *http://www.uclinug.org*. Le nombre d'architectures supportées a depuis explosé et µClinux se présente aujourd'hui comme un concurrent très sérieux des systèmes d'exploitation embarqués pour micro-contrôleurs. La liste des processeurs supportés par µClinux est disponible sur *http://www.uclinux.org/ports*. De même, la distribution des sources permet à des constructeurs de solutions matérielles de fournir des portages de µClinux pour leurs produits. C'est par exemple le cas du processeur NIOS développé par la société Altera (*http://www.altera.com*), pour lequel cette dernière fournit un environnement de développement complet basé sur µClinux et la chaîne de compilation GNU. La description de ce produit est disponible à l'adresse *http://www.altera.com/products/devkits/altera/kit-dev_linux.html*.

Le gros avantage de µClinux par rapport à ses concurrents est la compatibilité de l'API de programmation avec les systèmes Linux standards. Il dispose également de toutes les fonctionnalités réseau TCP/IP disponibles sur le noyau Linux, ce qui est également un très gros avantage dans le cas du développement d'un produit communicant. Le système µClinux est également très raisonnable au niveau de l'empreinte mémoire avec une taille de noyau inférieure à 512 Ko, la liste des commandes utilisateur étant inférieure à 900 Ko. Les micro-contrôleurs étant souvent utilisés dans des environnements temps réel, il existe également un portage de RTLinux pour le noyau µClinux 2.0 à l'adresse *http://www.uclinux.org/pub/uCsimm/Rt-port*.

L'API étant compatible avec celle de Linux, il est possible d'effectuer des portages d'applications vers l'environnement µClinux. De même, de nombreux outils et programmes adaptés à l'environnement réduit comme Boa (serveur http embarqué) sont disponibles pour µClinux.

L'absence de MMU impose cependant quelques contraintes par rapport à un environnement Linux classique :

- La mémoire virtuelle n'existe pas. Le noyau et les programmes applicatifs partagent le même espace d'adressage, ce qui implique qu'une application mal écrite peut mettre en péril le fonctionnement global du système.
- L'appel système `fork` n'est pas supporté. Il est remplacé par une implémentation de l'appel système `vfork` BSD (le processus parent est suspendu jusqu'à ce que le processus fils appelle `exec` ou `exit`).
- L'appel système `exec` ne peut pas charger une image binaire supérieure à 256 Ko.
- La taille de la pile est fixe pour chaque processus.

Quelques kits matériels disponibles

La mise en œuvre de µClinux est un peu plus complexe que celle d'un Linux classique car elle nécessite un environnement matériel particulier alors que Linux peut être testé sur une plate-forme x86 grand public. Pour utiliser µClinux, il est nécessaire de faire l'acquisition d'un kit d'évaluation (on peut aussi le fabriquer soi-même). Les kits d'évaluation uCsimm, Geek Kit, uCdimm et ARM7TDMI sont disponibles en ligne sur *http://www.uclinux.com*.

Le kit uCsimm

Le module uCsimm est conçu spécialement pour le système µClinux. Il est constitué d'un module SIMM standard 30 broches équipé d'un micro-contrôleur DragonBall M68EZ328 à 16 MHz, de 2 Mo de mémoire flash de 8 Mo de mémoire vive. Le module dispose également des fonctions suivantes :

- interface Ethernet 10BaseT,
- un port série RS-232,
- 21 entrées/sorties multi usages,
- pilote d'afficheur LCD ou QVGA,
- bus I2C ou SPI3 à 1 Mbit/s.

Le système est alimenté en 3,3 V. Il peut facilement être utilisé avec un connecteur SIMM et un câble série. La figure ci-après donne l'encombrement du module.

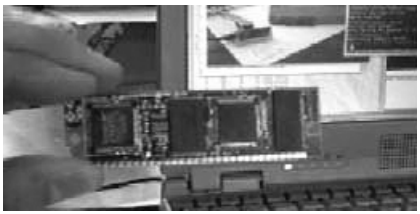


Figure 10-1
Module uCsimm.

Le kit uCgardener contient le module uCsimm, le CD-Rom µClinux et le kit de montage du support SIMM. La mémoire flash du module contient un moniteur permettant de télécharger en RAM l'image du système µClinux *via* le port série et de l'installer dans la mémoire flash. Ce kit existe en version économique, à monter soi-même, sous la référence Geek Kit.

Le kit uCdimm

Le kit uCdimm est une version plus performante du kit uCsimm. Il se présente sous la forme d'un module DIMM 144 broches de 1,7 sur 2,7 pouces. Ce dernier est équipé d'un micro-contrôleur DragonBall VZ (68VZ328) à 33 MHz, de 2 Mo de mémoire flash et 8 Mo de mémoire vive DRAM. Le module intègre en outre :

- un contrôleur de DRAM,
- deux ports série, deux interfaces SPI,
- un contrôleur LCD (résolution jusqu'à 640 × 512),
- jusqu'à 22 E/S parallèles,
- une horloge temps réel – calendrier,
- une interface Ethernet 10BaseT.

Le reste du contenu du kit est identique à celui du kit uCsimm.

Le kit ARM7TDMI

Basé sur un processeur ARM (ATMEL AT91), le kit ARM7TDMI offre plus de puissance que les modules précédents. Le kit est composé de la carte d'évaluation ATMEL EB40-LS et d'une carte fille Lineo µCnet pour la connexion Ethernet et l'extension de mémoire flash. Les caractéristiques du kit sont les suivantes :

- processeur ARM7TDMI avec 128 Ko de SRAM interne,
- 2 Mo de mémoire statique SRAM,
- 128 Ko de mémoire flash, plus 2 Mo sur la carte µcent,

- deux ports série,
- port JTAG,
- une interface Ethernet 10BaseT IEEE 802.3.

Le projet Open Hardware

Open Hardware (*http://www.openhardware.net*) a une approche similaire au développement Open Source mais appliquée au matériel. Tous les schémas, documentations et nomenclatures sont téléchargeables depuis le site Open Hardware. Le projet propose diverses cartes mère telle celle présentée sur la figure ci-après.

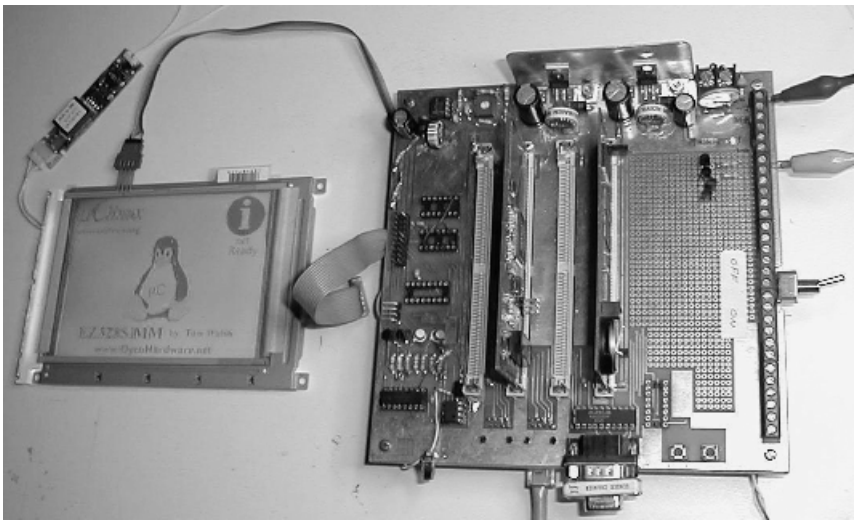


Figure 10-2
Exemple de carte Open Hardware.

Cette carte mère dispose de quatre emplacements SIMM 72 broches dont deux sont libres d'utilisation, plus un connecteur RJ45 pour l'accès Ethernet IEEE 802.3 10BaseT, une interface pour écran LCD et une interface pour écran tactile. Elle est équipée en plus :

- d'un processeur Motorola DragonBall EZ,
- de 8 Mo de DRAM,
- de 2, 4 ou 8 Mo de mémoire flash,
- d'une horloge temps réel avec sauvegarde par pile bouton,
- de 9 E/S parallèles,
- d'un module SIMM 72 broches comportant le contrôleur Ethernet.

La carte d'évaluation ColdFire Motorola M5407C3

La plate-forme matérielle utilisée dans la suite de ce chapitre est la carte d'évaluation Motorola M5407C3 à base de processeur ColdFire. Les cartes d'évaluation proposées par les fabricants de composants sont d'un coût peu élevé et représentent un moyen rapide de tester des fonctionnalités bien précises comme des convertisseurs analogiques/numériques, des amplificateurs opérationnels ou des processeurs.

La distribution µClinux pour ColdFire supporte un ensemble de cartes d'évaluation du commerce comme :

- les cartes Arnewsh SBC5206 et SBC5307,
- les cartes Motorola M5206eLITE, M5206C3, M5307C3, M5272C3,
- les cartes Lineo eLIA 5307, NETtel et SecureEdge,
- la carte Netburner CFV2-40.

La carte d'évaluation M5407C3 utilise le processeur ColdFire MCF5407 qui est le successeur du célèbre processeur Motorola 68000 dont la production s'est arrêtée avec le microprocesseur 68060 il y a quelques années. Il reprend le jeu d'instructions du microprocesseur 68020 dans lequel on a supprimé quelques instructions et modes d'adressage peu utilisés. Selon les versions de processeur ColdFire, Motorola a ajouté des instructions de type MAC (*Multiply and ACCumulate*) que l'on retrouve dans les processeurs de traitement du signal (*Digital Signal Processing* ou DSP). Une vue de la carte M5407C3 est présentée sur la figure ci-après.

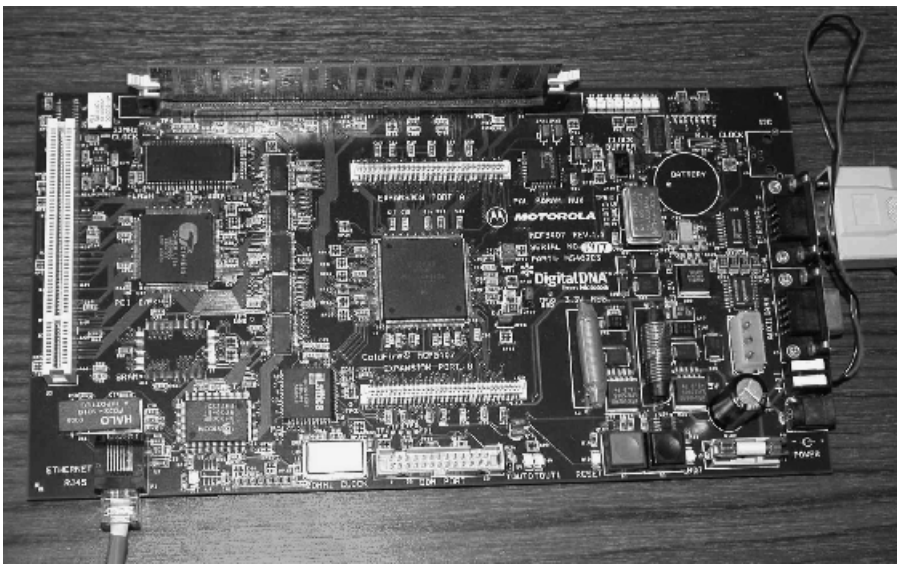


Figure 10-3
Carte M5407C3.

La carte d'évaluation M5407C3 possède :

- un processeur MCF5407 à 150 MHz,
- 32 Mo de mémoire SDRAM,
- 2 Mo de mémoire flash dont 0,3 Mo occupé par le moniteur de la carte,
- 2 ports série,
- une liaison Ethernet IEEE 802.3 10BaseT,
- un port de débogage BDM (*Background Debugger Module*),
- un emplacement PCI.

Mise en œuvre de µClinux

Nous utilisons pour la mise en œuvre de µClinux le processeur ColdFire qui est installé sur la carte d'évaluation Motorola M5407C3. La démarche à suivre serait bien sûr similaire pour des versions de µClinux adaptées à d'autres processeurs.

La distribution µClinux pour ColdFire est basée initialement sur un noyau Linux 2.0.38 mais propose aussi une version utilisant le noyau 2.4.10. L'ensemble est stable et offre une large palette de services comme :

- des clients NFS et SMB,
- un client DHCP,
- la fonction d'IP masquerading,
- des serveurs HTTP réduits (comme Boa et thttpd).

La mise en œuvre de µClinux/ColdFire pour la génération de l'image à télécharger dans la carte d'évaluation ressemble fort à une génération d'un noyau Linux. Il faut tout d'abord récupérer les archives sur le site µClinux/ColdFire, soit :

la chaîne de compilation croisée ColdFire (fichier `m68k-elf-tools-xxxxxxxx.tar.gz`),

la distribution de µClinux/ColdFire (fichier `uClinux-dist-xxxxxxxx.tar.gz`).

On doit tout d'abord installer sur le PC Linux hôte la chaîne de compilation croisée ColdFire (*cross development*). Pour ce faire, en tant que super-utilisateur, on exécute :

```
# cd /
# tar xvfz m68k-elf-tools-20010716.tar.gz
# exit
```

Il faut ensuite installer la distribution µClinux/ColdFire dans son répertoire de travail en tant que simple utilisateur.

```
$ tar xvfz uClinux-dist-20011112.tar.gz
$ cd uClinux-dist/
```

On retrouve ensuite l'enchaînement classique de compilation d'un noyau sous Linux, soit en premier lieu la partie configuration.

```
$ make xconfig
```

Le menu décrit ci-après permet de configurer le noyau µClinux.

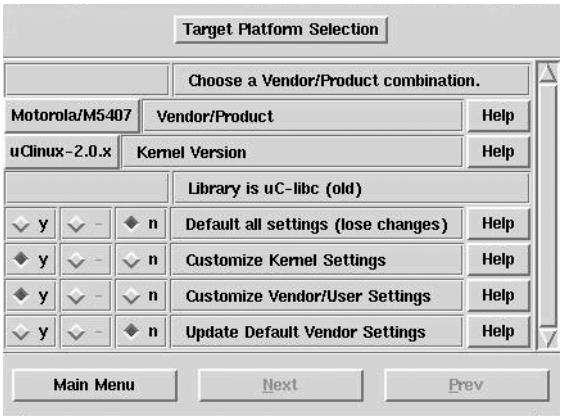


Figure 10-4

Configuration du noyau µClinux.

De même, le menu décrit ci-après permet de choisir et configurer les applications à utiliser. N'oublions pas que, dans µClinux, les applications et le noyau partagent le même espace d'adressage.

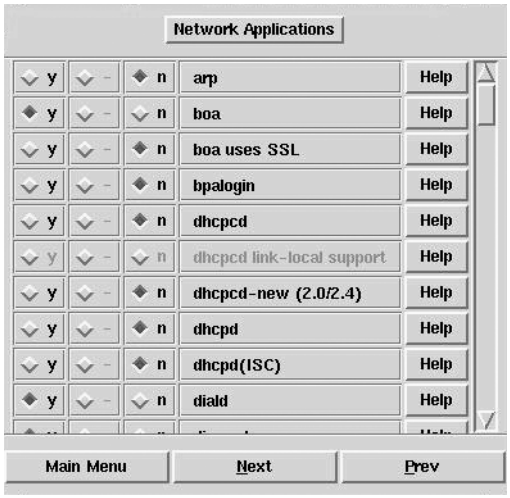


Figure 10-5

Configuration et choix des applications.

Après sauvegarde de la configuration, on effectue ensuite la compilation comme pour un noyau Linux. Notez que le compilateur utilisé n'est pas la version native de la station Linux mais la version croisée pour le processeur ColdFire.

```
$ make dep
$ make
```

Si la compilation se déroule sans erreur, l'image binaire **`image.bin`** du système µClinux/ColdFire est placée dans le répertoire **`/tftpboot`** afin d'être téléchargée dans la mémoire vive de la carte d'évaluation *via* le réseau Ethernet avec le protocole TFTP (*Trivial FTP*).

Pour ce faire, on se connecte à la carte par le port série en utilisant une application d'émulation de terminal comme **`kermit`** ou **`minicom`**, puis on utilise la commande **`dn`** du moniteur afin de démarrer le téléchargement.

```
$ kermit
Connecting to /dev/ttyS1, speed 19200.
The escape character is Ctrl-\ (ASCII 28, FS)
Type the escape character followed by C to get back,
or followed by ? to see other options.
----------------------------------------------------

Hard Reset
DRAM Size: 32M

Copyright 1995-1999 Motorola, Inc. All Rights Reserved.

ColdFire MCF5407 EVS Firmware v2e.1a.1a (Build 1 on Jul 27 2000 16:35:59)

Enter 'help' for help.

dBUG> dn image.bin
Eth Mac Addr is 00:CF:54:07:C3:01

Downloading Image 'image.bin' from 147.200.10.209
...................
1269540 bytes read via TFTP
```

Lorsque le téléchargement est terminé, on peut démarrer le noyau µClinux/ColdFire. Pour cela, on exécute :

```
dBUG> go 20000
uClinux/COLDFIRE(m5407)

COLDFIRE port done by Greg Ungerer, gerg@lineo.com

Flat model support (C) 1998,1999 Kenneth Albanowski, D. Jeff Dionne
Calibrating delay loop.. ok - 149.50 BogoMIPS

Memory available: 30892k/32768k RAM, 0k/0k ROM (359k kernel code, 140k data)
...
```

```
Execution Finished, Exiting
Sash command shell (version 1.1.1)
/>
```

On remarquera sur les traces précédentes que la taille du noyau est inférieure à 500 Ko. Nous disposons maintenant d'un système **Linux** embarqué opérationnel sur lequel on peut exécuter des applications, par exemple le serveur web embarqué Boa :

```
/> boa&
[20]
/>
```

Exemple d'application µClinux

L'exemple d'application simple présenté ici consiste en la mise en place d'une fonction de pilotage à distance d'un équipement électronique (en l'occurrence la carte d'évaluation) à travers un navigateur Internet en utilisant une technologie CGI (*Common Gateway Interface*). Pour cela, on utilise bien sûr le serveur Boa qui supporte l'interface CGI.

Nous rappelons ici que l'interface CGI est un moyen simple d'exécuter une application quelconque depuis une interface graphique HTML. Le principe met en œuvre la balise **FORM** du langage HTML de la manière suivante :

```
<BODY>
<FORM ACTION="led_control.cgi">
LED
<INPUT TYPE=radio NAME="CONTROLE" VALUE="on"> allumée
<INPUT TYPE=radio NAME="CONTROLE" VALUE="off"> éteinte
<P>
<INPUT TYPE=submit NAME="VALIDE" VALUE="Valider">
</FORM>
</BODY>
```

Le fichier **led_control.cgi** constitue le programme CGI et il peut être écrit dans n'importe quel langage à partir du moment où il respecte les spécifications de l'interface CGI. Pour tester notre environnement sur la station de travail, on peut pour l'instant utiliser le script shell suivant comme programme CGI.

```
#!/bin/sh
echo Content-Type: text/html
echo

echo "<BODY>QUERY_STRING= [$QUERY_STRING]</BODY>"
```

Les deux premières commandes **echo** constituent l'en-tête standard que doit envoyer le programme CGI au navigateur, le reste de la page est affiché dynamiquement grâce à la troisième commande **echo**. Par défaut, le programme CGI reçoit les paramètres de la page dans la variable d'environnement QUERY_STRING sous la forme d'une suite de chaînes de caractères séparées par le caractère &.

```
variable1=valeur1&variable2=valeur2&...&variableN=valeurN
```

Lorsque l'on visualise cette page avec un navigateur, on obtient l'écran suivant :

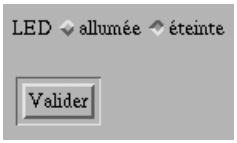


Figure 10-6
Test de la forme HTML.

Si l'on clique sur l'un ou l'autre des boutons (allumer ou éteindre la LED) puis que l'on clique sur **`Valider`**, on obtient l'affichage suivant dans la fenêtre du navigateur :

```
QUERY_STRING= [CONTROLE=off&VALIDE=Valider]
```

Dans le véritable programme CGI, il est donc simple d'extraire le paramètre de contrôle et de l'utiliser afin d'envoyer la bonne commande bas niveau pour éteindre ou allumer la LED.

```
#include <stdio.h>
#include <stdlib.h>

main ()
{
  char *param, control;
  extern char *getenv();
  short *PADAT = (short*)0x10000244;

  printf ("Content-Type: text/html\n\n");

  if (!(param = getenv ("QUERY_STRING"))) {
    printf ("<BODY><H1>Pas de paramètre!</H1></BODY>");
    exit (1);
  }

  /* Recherche de la valeur */
  while (*param && *param++ != '=')
    ;

  control = *param - '0';

  switch (control) {
  case 0:
    /* Eteint */
    printf ("<BODY><H1>LED éteinte</H1></BODY>");
    *PADAT |= 0x0080;
    break;
```

```
    case 1:
      /* Allumé */
      printf ("<BODY><H1>LED allumée</H1></BODY>");
      *PADAT &= 0x007f;
      break;

    default:
      printf ("<BODY><H1>Paramètre incorrect!</H1></BODY>", param);
      exit (1);
    }
  }
```

Dans le cas du système µClinux, le programme CGI est donc écrit en C (`ledctl.c`) et installé sur le répertoire `uClinux-dist/vendors/Generic/cgi`. Il est ensuite « *cross*-compilé », afin de générer le programme CGI (fichier `ledctl.cgi`) qui sera installé sur le répertoire `/home/httpd` de l'image µClinux. Le serveur Boa doit également être paramétré afin d'autoriser l'exécution de CGI.

En résumé

µClinux est une version de l'environnement Linux adaptée aux processeurs et micro-contrôleurs dépourvus de MMU. L'utilisation d'un système tel que Linux permet de profiter des fonctionnalités du système Linux standard au niveau réseau et services de type HTTP, fonctions qui sont souvent coûteuses ou difficiles à mettre en place dans les environnements propriétaires.

La mise en place d'une solution µClinux nécessite cependant un test préalable sur une carte d'évaluation, ce qui est un désavantage par rapport à une solution Linux classique qui peut être testée sur un système grand public à bas coût.

11

Développement croisé

Nous avons pour l'instant vu des exemples dans l'environnement Intel x86. Même si cet environnement est très répandu, il est loin d'être le plus utilisé dans les applications industrielles et embarquées. Comme nous l'avons vu au chapitre 3, d'autres processeurs comme l'ARM ou le PowerPC sont parfois mieux adaptés que l'architecture x86.

Cependant, la plupart des développeurs utiliseront un PC x86 (Linux ou Windows) comme poste de travail, il est donc nécessaire de mettre en place une chaîne de développement croisée permettant de produire du code non-x86 sur un PC. Dans ce chapitre, nous allons décrire plusieurs solutions Open Source disponibles, utilisables sur Linux x86. Nous expliquerons également comment mettre en œuvre certains de ces outils sur plate-forme Windows en utilisant l'environnement d'émulation CYGWIN.

À titre d'exemple, nous mettrons en place et testerons une chaîne de développement pour cible Linux ARM.

Principe de la compilation sous Linux

La chaîne de compilation GNU utilise les composants suivants :

- le compilateur qui constitue le paquetage `gcc` ;
- les outils annexes (assembleur, éditeur de liens, etc.) qui constituent le paquetage `binutils` ;
- la GNU-Libc qui constitue le paquetage `glibc`. Ce paquetage contient en général la bibliothèque de gestion des threads (LinuxThreads).

Dans le cas d'un système Linux x86, ces paquetages sont installés sur le système sous forme de paquetages RPM ou DEB dans le cas de la distribution Debian.

Dans le cas de la distribution Red Hat, nous aurons par exemple :

```
$ rpm -qv gcc
gcc-3.2.2-5
$ rpm -qv binutils
binutils-2.13.90.0.18-9
$ rpm -qv glibc
glibc-2.3.2-11.9
```

Concernant le paquetage **binutils**, ce dernier contient des outils utilisés pour la génération et la manipulation des exécutables ou des fichiers objets (fichiers .o) intermédiaires. On notera dans ce paquetage la présence de l'assembleur **as** ou de l'éditeur de liens **ld** :

```
$ rpm -ql binutils
/usr/bin/addr2line
/usr/bin/ar
/usr/bin/as
/usr/bin/gprof
/usr/bin/ld
/usr/bin/nm
/usr/bin/objcopy
/usr/bin/objdump
/usr/bin/ranlib
/usr/bin/readelf
/usr/bin/size
/usr/bin/strings
/usr/bin/strip
...
```

Dans le cas de la compilation croisée, ces différents outils, ainsi que le compilateur, seront exécutés dans un environnement x86 (Linux ou Windows) mais le code généré sera d'un type différent (par exemple ARM). Il est donc nécessaire de compiler le paquetage à partir des sources en spécifiant de nouvelles options de génération. Pour ce faire, le script **configure** inclus dans les paquetages permet de spécifier le type d'architecture sur lequel s'exécute l'outil (option **--host**) ainsi que le type d'architecture cible (option **--target**).

Dans le cas où les options ne sont pas précisées, le script générera par défaut une configuration pour un exécutable natif, à utiliser dans l'environnement de compilation courant. Voici un exemple pour le paquetage **binutils** :

```
$ ./configure
loading cache ./config.cache
checking host system type... i686-pc-linux-gnu
checking target system type... i686-pc-linux-gnu
checking build system type... i686-pc-linux-gnu
...
```

Par contre, si l'on précise les options citées précédemment on obtient :

```
$ ./configure --host=i686-pc-linux-gnu --target=arm-linux
creating cache ./config.cache
checking host system type... i686-pc-linux-gnu
checking target system type... arm-unknown-linux-gnu
checking build system type... i686-pc-linux-gnu
...
```

La génération de la chaîne de compilation croisée est donc réalisable à la main mais elle est souvent fastidieuse car elle peut nécessiter l'application de patches sur un ou plusieurs paquetages en fonction des différentes architectures. De ce fait, l'utilisateur non averti risque de disposer d'une chaîne erronée et il est donc conseillé d'utiliser des outils spécialisés.

Dans la suite du chapitre, nous présenterons donc l'outil ELDK (*Embedded Linux Development Kit*) développé par DENX Sofware (*http://www.denx.de/ELDK.html*) ainsi que l'outil CROSSTOOL développé par Dan Kegel (*http://kegel.com/crosstool*). Nous précisons que ces deux outils sont totalement libres et diffusés sous licence GPL. Il n'y a pas non plus de restrictions concernant les projets développés avec ces outils.

L'outil ELDK

ELDK est développé par Denx Sofware, société de consulting Linux située en Allemagne. Le fondateur, Wilfried Denk, est un professionnel de talent, contribuant à de nombreux projets Open Source dont RTAI (*http://www.rtai.org*).

Le produit est disponible sous forme de paquetages RPM binaires ou sources. Il est également possible de télécharger l'image ISO du CD-Rom contenant les binaires ou les sources. ELDK permet de mettre en place une chaîne de compilation utilisable sur un PC Linux x86 pour des cibles PowerPC, MIPS ou ARM. Cet outil fournit également une distribution que l'on peut utiliser comme *root-filesystem* au travers d'un montage NFS. Cette technique facilite la mise au point car il n'est pas nécessaire de mettre à jour systématiquement la cible. La version de noyau fournie par ELDK est la 2.4.25 améliorée par DENX principalement pour la plate-forme PowerPC.

L'installation est très simple. Il suffit de télécharger l'image ISO du CD-Rom auprès d'un des miroirs du projet, soit :

- *ftp://mirror.switch.ch/mirror/eldk/eldk/*
- *http://mirror.switch.ch/ftp/mirror/eldk/eldk/*
- *ftp://sunsite.utk.edu/pub/linux/eldk/*
- *http://sunsite.utk.edu/ftp/pub/linux/eldk/*
- *ftp://ftp.sunet.se/pub/Linux/distributions/eldk/*
- *http://ftp.sunet.se/pub/Linux/distributions/eldk/*
- *ftp://ftp.leo.org/pub/eldk/*
- *http://archiv.leo.org/pub/comp/os/unix/linux/eldk/*

La dernière version à ce jour est la 3.1.1 et il faut noter que les images ISO n'existent pas forcément pour les anciennes versions comme la 2.1.0. À partir de l'image ISO, il est aisé de graver le CD-Rom sur un système Linux, ou même sur une machine Windows. On peut aussi utiliser l'image ISO grâce au *loopback device* décrit au chapitre 5 et monter le fichier en utilisant la commande suivante :

```
mount -t iso9660 -o loop mon_image.iso /mnt/cdrom
```

Lorsque l'image (ou le CD-Rom) est montée, on peut installer la distribution en exécutant simplement la commande suivante :

```
$ /mnt/cdrom/install -d /opt/eldk_arm
Do you really want to install into /opt/eldk_arm directory[y/n]?: y

Creating directories
Done
Installing cross RPMs

Preparing...           ########################################### [100%]
   1:rpm               ########################################### [100%]
Preparing...           ########################################### [100%]
   1:rpm-build         ########################################### [100%]
Preparing...           ########################################### [100%]
   1:binutils-arm      ########################################### [100%]
...
```

Dans le cas du PowerPC, on pourra préciser les architectures à installer (ppc_4xx, ppc_6xx, etc.). Par défaut, la procédure installe toutes les architectures.

```
$ /mnt/cdrom/install -d /opt/eldk_ppc ppc_4xx ppc_6xx
```

Remarque

Il n'est pas nécessaire d'installer ELDK en tant que super-utilisateur (root). Dans le cas où ELDK est installé en tant que simple utilisateur, il est bien entendu nécessaire d'avoir les droits d'écriture sur le répertoire d'installation. Dans le cas de l'utilisation du root-filesystem ELDK par NFS, il sera nécessaire d'utiliser le script `ELDK_MAKEDEV` pour créer les entrées dans le répertoire **`/dev`** de l'image fournie par ELDK. Si ELDK n'est pas installé en tant que super-utilisateur, il faudra utiliser le script `ELDK_FIXOWNER` pour configurer correctement l'image NFS comme décrit dans la documentation ELDK sur *http://www.denx.de/twiki/bin/view/DULG/ELDKMountingTargetComponentsViaNFS*.

Lorsque le paquetage est installé, la chaîne de compilation est utilisable si l'on rajoute le chemin d'accès aux outils à sa variable d'environnement PATH comme ci-dessous :

```
$ PATH=/opt/eldk_arm/usr/bin:$PATH
$ export PATH
```

À partir de là, on peut utiliser le compilateur et les outils associés :

```
$ arm-linux-gcc -v
Lecture des spécifications à partir de /opt/eldk_arm/usr/bin/../lib/gcc-lib/
➥arm-linux/3.3.3/specs
Configuré avec: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/
➥share/info --enable-shared --enable-threads=posix --disable-checking --with-system-
➥zlib --enable-__cxa_atexit --with-newlib --enable-languages=c,c++ --disable-libgcj
➥--host=i386-redhat-linux --target=arm-linux
Modèle de thread: posix
version gcc 3.3.3 (DENX ELDK 3.1.1 3.3.3-9)
```

On peut compiler le programme de test traditionnel *Hello World* :

```
$ arm-linux-gcc -o helloword helloworld.c
$ file helloworld
helloworld: ELF 32-bit LSB executable, ARM, version 1 (ARM), for GNU/Linux 2.2.5,
➥dynamically linked (uses shared libs), not stripped
```

La dernière commande indique bien que nous sommes en présence d'un exécutable ARM.

L'outil CROSSTOOL

L'outil CROSSTOOL est quelque peu différent car il est plus complexe à appréhender mais aussi plus souple. Le complexité vient du fait qu'il n'existe pas de distribution binaire ni de programme d'installation aussi simple que pour ELDK. C'est d'ailleurs tout à fait normal car le but de CROSSTOOL est de fournir à l'utilisateur un ensemble de scripts lui permettant de construire sa chaîne de compilation même dans les cas les plus complexes alors qu'ELDK est limité à l'hôte Linux x86 et aux cibles PowerPC, MIPS et ARM. Avec CROSSTOOL, on choisit donc l'environnement hôte (host), la cible (target), les versions des paquetages à utiliser (gcc, binutils, etc.) mais aussi les différents patches à appliquer aux différents paquetages ou bien le noyau utilisé et ses patches associés. CROSSTOOL effectue également le téléchargement des paquetages nécessaires à la génération de la chaîne définie par l'utilisateur.

L'outil CROSTOOL est entièrement écrit en langage script shell ce qui lui donne un air *old style* qui n'est pas pour déplaire aux *geeks* chevronnés.

Définition récréative

Le terme *geek* vient de l'argot des informaticiens anglo-saxons et désigne un passionné de technique informatique plus enclin à l'utilisation des obscurs langages de script, ésotériques mais efficaces que des interfaces graphiques et autres « cliquodromes ». Le geek a une fâcheuse tendance à une certaine goujaterie voire un certain machisme involontaire qui est plus un effet de bord dû à sa passion qu'un véritable choix de comportement. Démontrant une fois de plus le sens pratique féminin, certaines compagnes de *geeks* se sont associées en France sous le terme « copines de geek » (*http://www.copinedegeek.com*).

Pour utiliser CROSSTOOL, il faut tout d'abord installer l'archive téléchargée depuis le site de Dan Kegel. Dans notre cas nous allons utiliser la version 0.31, dernière à jour au moment de l'écriture de ces lignes. Une fois la distribution installée, une documentation en anglais est disponible dans le fichier `doc/crosstool-howto.html`.

La structure du répertoire `crosstool-0.31` est très simple, les fichiers importants étant localisés directement sur la racine du répertoire :

- Le fichier `all.sh` est le script principal de génération de la chaîne.
- Les fichiers type `demo-xxx.sh` comme `demo-i686.sh` sont des exemples fournis dont l'utilisateur peut s'inspirer pour construire sa propre configuration.

- Les fichiers **.dat** permettent de définir des variables d'environnement utilisées par les scripts. Par exemple, le fichier **i686-cygwin.dat** définit des variables indiquant que la cible est CYGWIN.

```
$ cat i686-cygwin.dat
TARGET=i686-pc-cygwin
TARGET_CFLAGS="-O"
```

- Les fichiers **.config** correspondent à des configurations du noyau Linux générées par un **make xconfig** ou un **make config** et ce pour les différents processeurs supportés (comme **i686.config**, **m68k.config**). Ce principe peut être étendu à tout autre système de configuration utilisant un format similaire. Ces fichiers sont utilisés par les fichiers **.dat** cités précédemment.

```
$ cat i686.dat
KERNELCONFIG=`pwd`/i686.config
TARGET=i686-unknown-linux-gnu
TARGET_CFLAGS="-O"
```

D'autres sous-répertoires sont présents et nous pouvons citer :

- Le répertoire **download** qui accueille les paquetages sources manquants téléchargés par CROSSTOOL lors de la génération de la chaîne.
- Le répertoire **patches** qui contient lui-même des sous-répertoires correspondant aux différents paquetages utilisables. Chaque sous-répertoire contient les patches à appliquer en fonction de la configuration définie par l'utilisateur. Nous pouvons donner l'exemple du paquetage correspondant à **gcc-3.3.4** :

```
$ ls -w 1 patches/gcc-3.3.4/
gcc-3.3.4-arm-bigendian.patch
gcc-3.3.4-libstdcxx-sh.patch
gcc-3.3.4-ppc-asm-spec.patch
```

La documentation propose en tant qu'introduction de générer une chaîne de compilation native i686 ce qui n'a pas beaucoup d'intérêt autre que pédagogique. Pour cela, il suffit d'utiliser la commande suivante :

```
$ ./demo-i686.sh
```

Le script est extrêmement simple, nous avons au début du fichier les lignes suivantes :

```
#!/bin/sh
set -ex
TARBALLS_DIR=$HOME/downloads
RESULT_TOP=/opt/crosstool
export TARBALLS_DIR RESULT_TOP
GCC_LANGUAGES="c,c++"
export GCC_LANGUAGES

# Really, you should do the mkdir before running this,
# and chown /opt/crosstool to yourself so you don't need to run as root.
mkdir -p $RESULT_TOP
```

Suite à cela, la génération de la chaîne est lancée par la ligne suivante, qui indique que la chaîne en question sera basée sur la configuration i686, le compilateur **gcc-3.4.3** et la **glibc-2.3.4** :

```
eval 'cat i686.dat gcc-3.4.3-glibc-2.3.4.dat' sh all.sh -notest
```

Ce petit exemple n'est cependant pas très significatif et nous allons construire un script `demo-at91.sh` permettant de créer une chaîne de compilation croisée Linux x86 vers Linux ARM ATMEL AT91RM9200. Cette chaîne est similaire à celle fournie dans ELDK.

Au début du fichier, nous indiquons le répertoire dans lequel CROSSTOOL stockera les paquetages téléchargés :

```
TARBALLS_DIR='pwd'/downloads
export TARBALLS_DIR
mkdir -p $TARBALLS_DIR
```

Ensuite, nous pouvons définir le répertoire destination de la cible :

```
RESULT_TOP=/opt/crosstool
export RESULT_TOP
mkdir -p $RESULT_TOP
```

Comme pour l'exemple précédent, la chaîne pourra traiter les langages C et C++ :

```
GCC_LANGUAGES="c,c++"
export GCC_LANGUAGES
```

La chaîne utilisera le noyau 2.4.27, la configuration du noyau étant définie dans le fichier **config-kernel** :

```
KERNELCONFIG='pwd'/patches/linux-2.4.27/config-kernel
export KERNELCONFIG
```

Le répertoire **linux-2.4.27** n'existe pas dans la distribution CROSSTOOL mais il suffit de l'ajouter. Le répertoire contient les différents patches à appliquer au noyau ainsi que le fichier de configuration du noyau :

```
$ cd patches/linux-2.4.27
$ ls -w 1
01_2.4.27-vrs1.patch
02_2.4.27-vrs1-at91-06102004.patch
03_2.4.27-mkdep+cross-depmod.patch
04_2.4.27-kbd+sram+flash.patch
config-kernel
```

Remarque

Pour que CROSSTOOL considère le fichier comme un patch, le nom du fichier doit obligatoirement contenir la chaîne de caractère **patch** ou bien le suffixe `.diff`. De même, il est conseillé de numéroter les fichiers de patch en commençant leur nom par l'ordre d'application (01, 02, etc.). Le lecteur désirant plus d'information pourra consulter le code source du script `getandpatch.sh`.

Pour revenir à notre script principal, la cible de la chaîne est l'architecture **arm-linux** :

```
TARGET=arm-linux
export TARGET
TARGET_CFLAGS="-O"
export TARGET_CFLAGS
```

La chaîne de compilation correspondante sera installée dans **/opt/crosstool/arm** :

```
PREFIX=${RESULT_TOP}/arm
export PREFIX
```

Enfin, nous définissons la liste des paquetages à utiliser :

```
BINUTILS_DIR=binutils-2.15
GCC_DIR=gcc-3.3.4
GLIBC_DIR=glibc-2.3.2
GLIBCTHREADS_FILENAME=glibc-linuxthreads-2.3.2
LINUX_DIR=linux-2.4.27
export BINUTILS_DIR GCC_DIR GLIBC_DIR GLIBCTHREADS_FILENAME LINUX_DIR
```

La génération de la chaîne elle-même est effectuée par la dernière ligne :

```
eval sh all.sh --notest
```

Après avoir lancé la génération par la commande **demo-at91.sh**, nous devons attendre de longues heures avant de récolter le fruit de nos efforts. Afin de suivre le déroulement de l'exécution, il est souhaitable de conserver les traces dans un fichier par la commande :

```
$ ./demo-at91.sh 1>build.log 2>&1 &
```

On peut de temps en temps suivre l'évolution de la compilation par la commande :

```
$ tail -f build.log
```

Lorsque la chaîne est générée avec succès nous obtenons le message suivant dans le fichier :

```
Cross-toolchain build complete. Result in /opt/crosstool/arm.
testhello: C compiler can in fact build a trivial program.
Done.
```

À partir de là, on peut tester la chaîne de compilation comme nous l'avons fait pour ELDK :

```
$ PATH=/opt/crosstool/arm/bin:$PATH
$ export PATH
$ arm-linux-gcc -v
Reading specs from /opt/crosstool/arm/lib/gcc-lib/arm-linux/3.3.4/specs
Configured with: /home/pierre/test/crosstool-0.31/build/arm-linux/gcc-3.3.4-glibc-
2.3.2/gcc-3.3.4/configure --target=arm-linux --host=i686-host_pc-linux-gnu -prefix=/
opt/crosstool/arm --with-headers=/opt/crosstool/arm/arm-linux/include --with-local-
prefix=/opt/crosstool/arm/arm-linux --disable-nls --enable-threads=posix --enable-
symvers=gnu --enable-__cxa_atexit --enable-languages=c,c++ --enable-shared --enable-
c99 --enable-long-long
Thread model: posix
gcc version 3.3.4
```

On peut là aussi compiler le programme de test *Hello World :*

```
$ arm-linux-gcc -o helloword helloworld.c
$ file helloworld
helloworld: ELF 32-bit LSB executable, ARM, version 1 (ARM), for GNU/Linux 2.4.3,
➥dynamically linked (uses shared libs), not stripped
```

Utilisation de l'environnement CYGWIN

La quasi-totalité des outils présentés dans cet ouvrage est utilisable dans un environnement Linux. Cependant, il est clair que c'est loin d'être aujourd'hui l'environnement de développement le plus utilisé. La plupart des outils commerciaux sont souvent disponibles exclusivement sous Windows et la mise en place d'une chaîne de développement sous Linux peut poser des problèmes dans les grandes entreprises pour lesquelles la gestion du parc informatique est centralisée.

La société CYGNUS (liée à Red Hat Software) propose depuis longtemps un environnement sous Windows permettant de porter très rapidement les applications de Linux vers Windows. Le principe est de diriger les appels système normalement destinés au noyau Linux vers une DLL (pour *Dynamic Loading Library*) Windows fournie par CYGWIN, soit `CYGWIN.DLL`.

La majorité des outils disponibles sous Linux sont de ce fait disponibles sous CYGWIN, citons entre-autres :

- les commandes Linux, à commencer par l'interpréteur de commande `bash` ;
- la chaîne de compilation GNU ;
- l'environnement graphique XFree86 ;
- le bureau graphique KDE.

De ce fait, il est possible à l'aide de CROSSTOOL de construire une chaîne de compilation croisée utilisable sous Windows et CYGWIN et permettant de générer du code ARM.

Remarque

À matériel équivalent, il faut cependant noter que la configuration CYGWIN sera moins performante que la même chaîne utilisée sur un système Linux. De même, il existe encore quelques problèmes dans les adaptations CYGWIN des environnements graphiques comme KDE. La solution CYGWIN est donc à utiliser en dernier recours.

Installation de l'environnement CYGWIN

La distribution CYGWIN est utilisable sur des environnement Windows récents comme Windows 2000 ou Windows XP. Elle est disponible sur Internet sur le site *http://www.cygwin.com*. À partir de la page d'accueil du site, on peut télécharger le fichier `setup.exe` qui est une application Windows permettant d'installer la suite de la distribution.

Remarque

Il est préférable d'installer CYGWIN sur une partition de type NTFS et non VFAT afin d'éviter certains problèmes de droits d'accès aux fichiers. On peut connaître le type de système de fichiers en affichant les propriétés Windows de la partition en question (utiliser le bouton droit de la souris sur le volume correspondant).

Lorsque l'on double-clique sur l'icône associée au fichier `setup.exe`, on obtient la fenêtre suivante :

Figure 11-1

Exécution de setup.exe.

Lors de la première installation, il est conseillé d'utiliser l'option *Install from Internet.* Les fichiers utilisés sont sauvés sur le disque local ce qui permettra d'effectuer ultérieurement une installation à partir d'un répertoire local (*Install from Local Directory*) si cela était nécessaire.

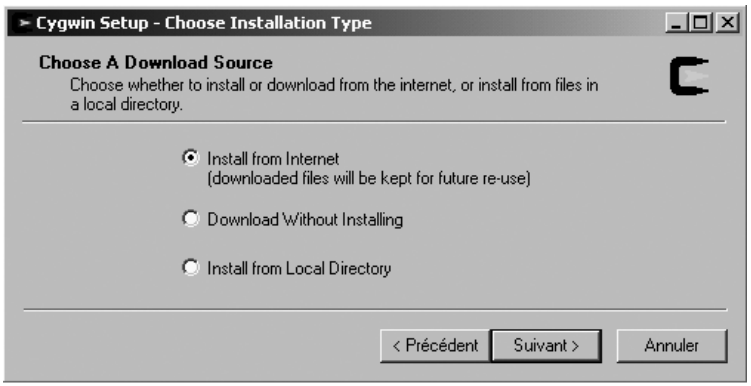


Figure 11-2

Choix du type d'installation.

L'écran suivant permet de sélectionner le répertoire d'installation (soit `c:\cygwin` par défaut) :

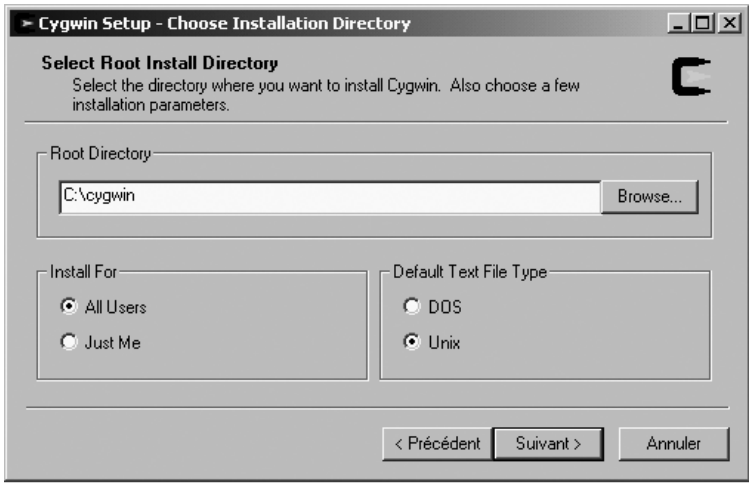


Figure 11-3
Choix du répertoire d'installation.

On sélectionne ensuite un serveur miroir de la distribution CYGWIN permettant le chargement des fichiers :

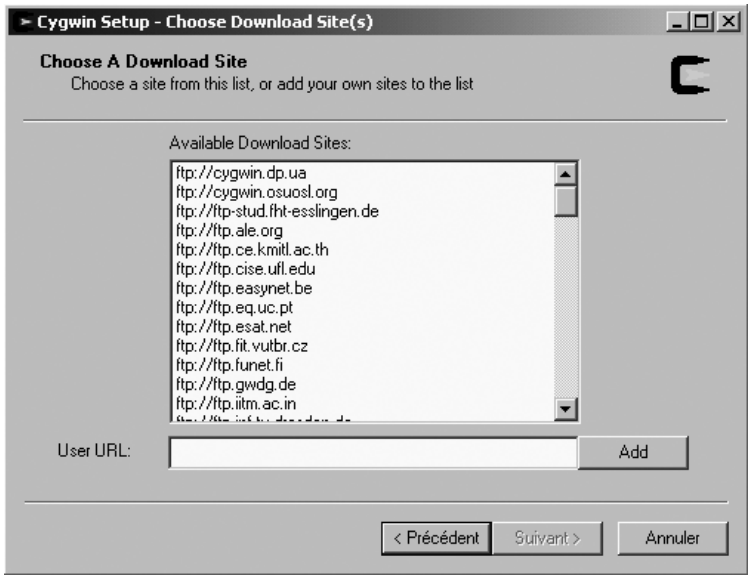


Figure 11-4
Choix du serveur miroir.

Le programme d'installation récupère alors la liste des paquetages et la présente à l'utilisateur. Dans le cas présent nous conseillons d'installer la totalité de la distribution :

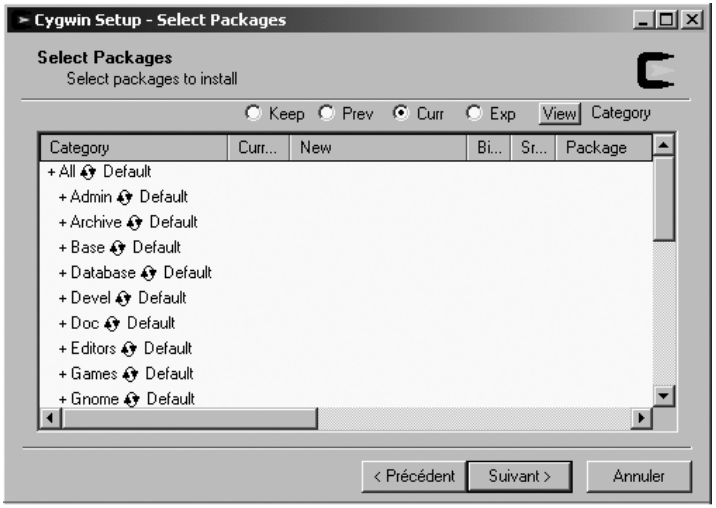


Figure 11-5

Liste des paquetages.

Pour ce faire, il faut cliquer sur le mot *Default* situé sur la première ligne de la liste (commençant par *All*). Le programme affiche alors *Install* à la place de *Default* dans toute la liste, ce qui indique que tous les paquetages sont installés. Il faut noter que l'installation de tous les paquetages occupe plus de 2 Go sur le disque.

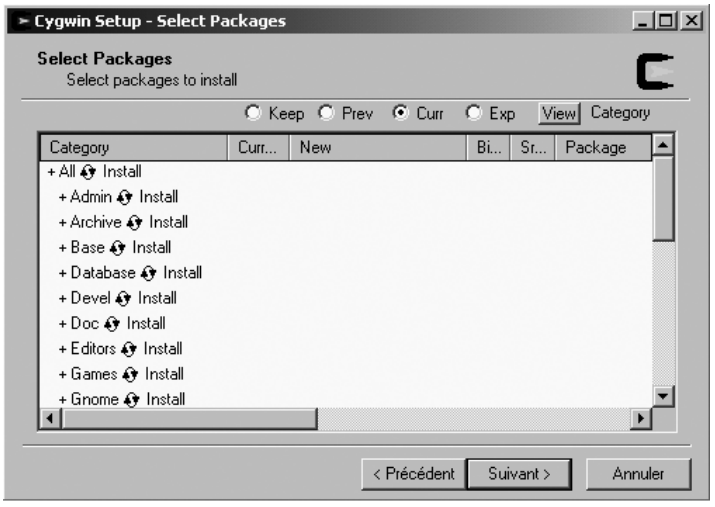


Figure 11-6

Sélection des paquetages.

Si l'on clique sur *Suivant*, l'installation doit démarrer. En cas de blocage de l'installation à cause d'un problème d'accès réseau, il est possible d'interrompre le programme d'installation puis de l'exécuter de nouveau. L'installation reprendra au niveau du dernier paquetage installé.

Lorsque l'installation est terminée, on doit obtenir une icône Cygwin sur le bureau Windows, ce qui permet d'ouvrir le terminal CYGWIN qui a une fière allure de bon vieux terminal Linux :

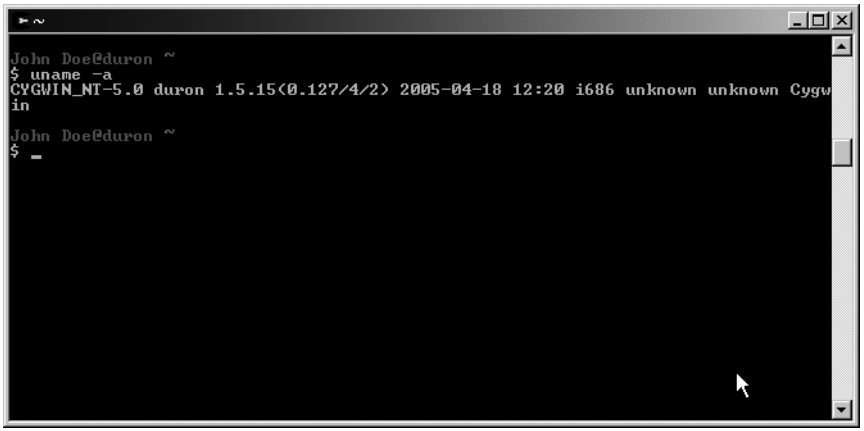


Figure 11-7
Terminal CYGWIN.

En premier lieu, il faut exécuter la commande `rebaseall -v` dans le terminal. Cette commande est nécessaire car elle permet d'affecter une adresse de base unique aux différentes DLL et donc permettre un chargement ordonné des bibliothèques.

On peut ensuite démarrer l'environnement XFree86 qui par défaut affiche un émulateur de terminal `xterm`. À partir de ce terminal, on peut lancer les commandes Linux habituelles comme le montre la figure 11-8.

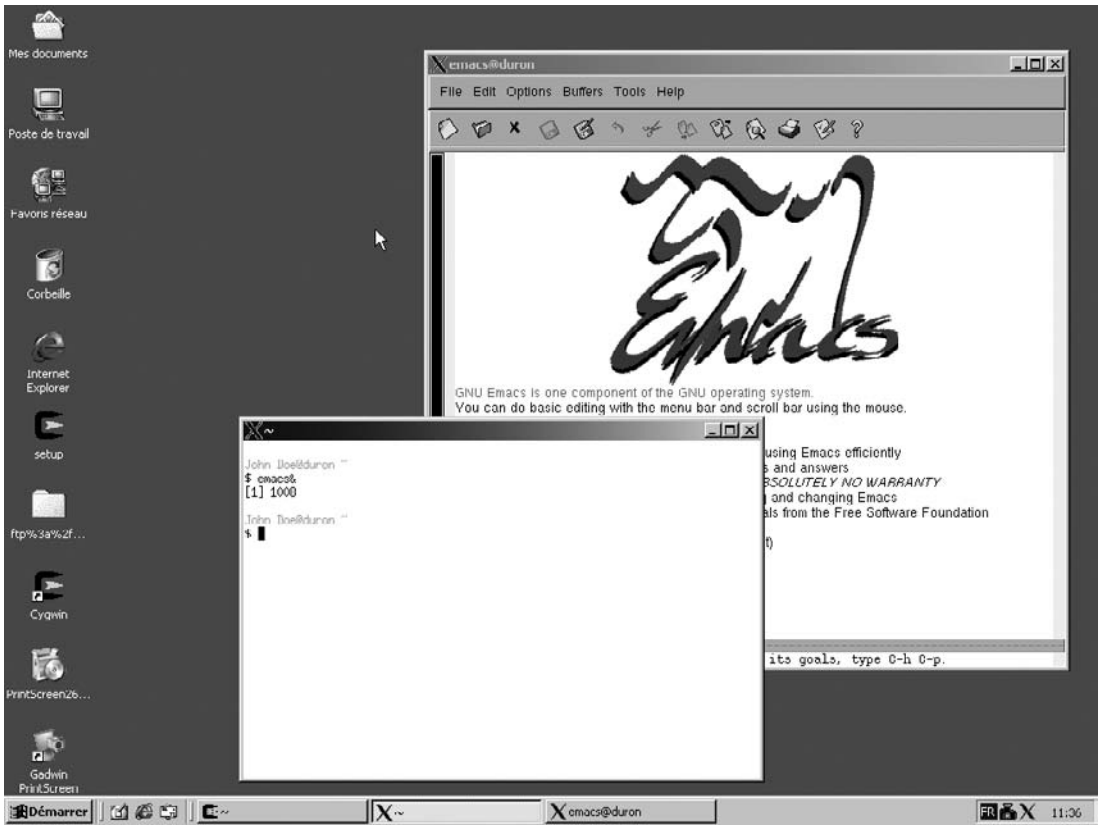


Figure 11-8
Environnement CYGWIN.

Création de la chaîne de compilation croisée pour ARM

L'environnement Linux quasiment complet disponible, la procédure de génération de la chaîne croisée CROSSTOOL est identique à celle utilisée sous Linux, décrite plus haut dans ce même chapitre. Il faut noter cependant que le type de plate-forme de développement détecté par le script de configuration `configure` est maintenant `i686-host_pc-cygwin` au lieu de `i686-pc-linux-gnu`.

Attention

Les scripts de CROSSTOOL ne s'entendent pas forcément très bien avec les noms de répertoires contenant des espaces, ce qui est fréquent sous Windows. Il est donc préférable d'extraire l'archive CROSSTOOL dans un répertoire ayant un nom UNIX comme `/home/test` et non pas `/home/mon répertoire`.

Après quelques heures de compilation (!), on obtient la même chaîne de développement contenant le compilateur **arm-linux-gcc**. On peut y accéder de la même manière que sous Linux :

```
$ export PATH=/opt/crosstool/arm/bin:$PATH
$ arm-linux-gcc -v
Reading specs from /opt/crosstool/arm/lib/gcc-lib/arm-linux/3.3.4/specs
Configured with: /home/test/crosstool-0.31/build/arm-linux/gcc-3.3.4-glibc-2.3.2/
gcc-3.3.4/configure --target=arm-linux --host=i686-host_pc-cygwin --prefix=/opt/
crosstool/arm --with-headers=/opt/crosstool/arm/arm-linux/include --with-local-
prefix=/opt/crosstool/arm/arm-linux --disable-nls --enable-threads=posix --enable-
symvers=gnu --enable-__cxa_atexit --enable-languages=c,c++ --enable-shared --enable-
c99 --enable-long-long
Thread model: posix
gcc version 3.3.4
```

Et l'on peut bien sûr compiler l'exemple de la même manière :

```
$ pwd
/home/John Doe
$ arm-linux-gcc -o helloworld helloworld.c
$ file helloworld
helloworld: ELF 32-bit LSB executable, ARM, version 1 (ARM), for GNU/Linux 2.4.3,
➥dynamically linked (uses shared libs), not stripped
```

Exemple de compilation

Lorsqu'une chaîne de compilation est installée, il est possible de construire des outils à partir des paquetages. Voici quelques exemples, dont un noyau Linux utilisable sur architecture ARM.

Compilation d'un noyau Linux ARM/AT91RM9200

L'architecture ARM est supportée par le noyau Linux mais il est nécessaire d'appliquer des patches disponibles sur *http://www.arm.linux.org.uk/developer*.

Dans le cas d'un noyau 2.4.27, on doit tout d'abord extraire l'archive du noyau standard (voir chapitre 4) puis appliquer le patch générique ARM puis le patch spécifique au composant AT91RM9200 :

```
# cd linux-2.4.27
# patch -p1 < 2.4.27-vrs1.patch
# patch -p1 < ../linux-patch/02_2.4.27-vrs1-at91-06102004.patch
```

La suite est très similaire à la compilation d'un noyau natif sauf que l'on doit utiliser les variables d'environnements **ARCH** et **CROSS_COMPILE** pour spécifier la chaîne de développement croisée. Pour configurer le noyau on utilisera la commande suivante :

```
# make ARCH=arm CROSS_COMPILE=arm-linux- menuconfig
```

Pour le compiler on utilisera la commande suivante :

```
# make ARCH=arm CROSS_COMPILE=arm-linux- dep zImage modules
```

Enfin, pour installer les modules sur un répertoire image de la cible, on utilisera la commande suivante :

```
# make ARCH=arm CROSS_COMPILE=arm-linux- INSTALL_MOD_PATH=/target_arm modules_install
```

Programme gdbserver

Cet outil a été décrit au chapitre 7. Il permet de déboguer un programme exécuté sur la cible depuis le poste de développement à travers un lien réseau ou RS-232. Pour générer la version ARM on utilisera les commandes suivantes dans le répertoire des sources de **`gdbserver`** :

```
$ ./configure --build=i686-pc-linux-gnu --host=arm-linux
$ make
```

Débogueur GDB croisé

Un tel outil permettra de déboguer à distance un programme exécuté sur une cible ARM via l'outil précédent :

```
$ ./configure --prefix=/opt/crosstool/arm --target=arm-linux --program-prefix=arm-linux-
$ make
```

Débogueur GDB natif ARM

La commande suivante permet de construire une version de **`gdb`** directement utilisable sur la cible ARM :

```
$ ./configure -build=i686-pc-linux-gnu --host=arm-linux --prefix=/usr/local/bin
$ make
```

Bibliothèque NCURSES native ARM

Nous construisons ici une version ARM de la bibliothèque NCURSES qui pourra être ajoutée à la chaîne de compilation :

```
$ BUILD_CC=gcc CC=arm-linux-gcc configure -build=i686-pc-linux-gnu --host=arm-linux
➥--prefix=/opt/crosstool/arm/arm-linux --with-shared
$ make
```

Résumé

La création de la chaîne de compilation est une étape essentielle de la mise en place de l'environnement de développement en cas d'utilisation d'une cible d'architecture différente. La construction entièrement manuelle d'une chaîne de compilation croisée est fastidieuse.

Pour certaines architectures, il existe des chaînes de compilation croisées au format binaire et très faciles à installer (ELDK) mais il est possible de construire sa propre chaîne à l'aide de l'outil CROSSTOOL.

L'environnement CYGWIN permet d'utiliser les outils Linux sous Windows 2000 ou XP. Les performances sont inférieures à celle de Linux mais l'utilisation de CROSSTOOL est strictement identique à celles de Linux. Il est donc très simple de construire une chaîne de développement croisée utilisable sous Windows/Cygwin.

12

Interfaces graphiques

Mode texte (console standard)

L'interface graphique est souvent un problème mineur dans le cas d'une application embarquée. Bien des systèmes embarqués n'utilisent pas de console au sens PC du terme (un écran et un clavier). Dans le chapitre 5, on a d'ailleurs montré comment on peut configurer le noyau Linux et LILO pour rediriger la console du système sur un port série. Dans ce type de système, l'interface utilisateur est réduite aux actions de maintenance *via* une connexion locale sur la console, ou distante à travers un réseau, et les protocoles Telnet, SSH ou FTP.

Le paquetage *console-tools* inclut des outils de manipulation du clavier et de l'écran de la console Linux. Ce paquetage contient entre autres l'utilitaire **`loadkeys`** et les fichiers de configuration des différents claviers nationaux. Cet utilitaire a déjà été utilisé au chapitre 6 pour configurer le clavier français. L'adaptation d'un nouveau clavier passe par la création d'un fichier *map* (carte) permettant de faire le lien entre le code matériel envoyé par la touche (*scancode* ou *keycode*) et le code ASCII à afficher dans la police de caractères choisie. Le programme **`showkey`** permet de connaître les codes des touches.

```
# showkey
press any key (program terminates after 10s of last keypress)...
keycode 28 released
keycode 16 press
keycode 16 release
```

Le programme **`dumpkeys`** permet de visualiser la carte du clavier utilisée par **`loadkeys`**. Une fois la carte stockée dans un fichier, on peut l'adapter au clavier réel puis utiliser la nouvelle carte avec **`loadkeys`**.

```
# dumpkeys > monclavier
# cp monclavier nouveauclavier
```

Après modification du fichier **nouveau_clavier** en fonction des valeurs de codes obtenues par **showkeys**, on peut exécuter :

```
# loadkeys nouveauclavier
```

On peut également modifier la police de caractères de la console en utilisant la commande **consolechars**. Cette modification n'est pas qu'esthétique et peut se révéler nécessaire dans le cas de langues n'utilisant pas l'alphabet latin. Ainsi, pour passer à une police en alphabet cyrillique, on pourra faire :

```
# consolechars -f UniCyr_8x14
```

et, pour revenir à la police par défaut :

```
# consolechars -d
```

Les fichiers correspondant aux polices de caractères sont situés sur **/lib/kbd/consolefonts**. Nous ne donnons ici, bien entendu, qu'un petit aperçu des manipulations possibles et le lecteur plus exigeant pourra se référer au document *Keyboard-and-Console-HOWTO* disponible sur *http://www.linuxdoc.org*.

X Window System

Une introduction à X

X Window System est traditionnellement l'environnement graphique de prédilection pour les systèmes de type Unix. Conçu dans les années 1980 dans les laboratoires du MIT (Massachusetts Institute of Technology), sa vocation initiale était de constituer un système graphique « réparti » constitué de terminaux graphiques banalisés (ou DISPLAY) équipés d'un logiciel appelé « serveur X ». Le serveur X reçoit à travers le réseau des requêtes graphiques des « clients X » que sont les programmes applicatifs exécutés sur des systèmes distants. Le serveur X et les clients peuvent bien entendu être situés sur la même machine physique, ce qui est très souvent le cas.

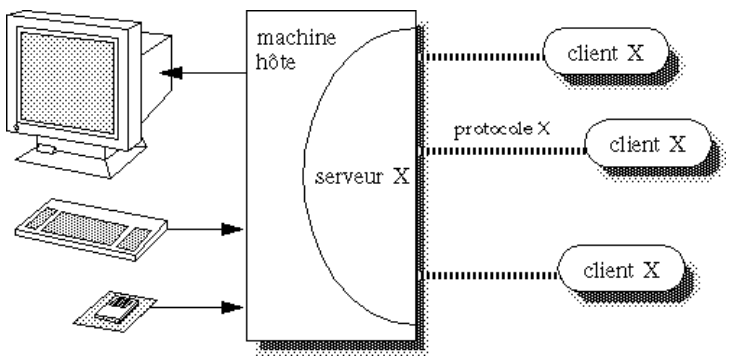


Figure 12-1
Architecture X Window System.

Le terminal ou DISPLAY est constitué d'un trio écran/clavier/souris. La conception du système était très novatrice à l'époque car les architectures matérielles et logicielles tant du terminal X recevant les requête graphiques que du serveur exécutant les applications peuvent être très différentes. De même, le support de transport du protocole X est banalisé, bien qu'il soit souvent basé sur une connexion TCP/IP. Au niveau du client, l'accès aux différentes couches de X est assuré par des bibliothèques, comme cela est indiqué sur la figure ci-après :

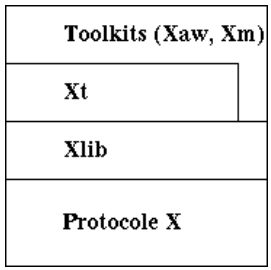


Figure 12-2
Les bibliothèques X.

Le protocole X est défini sous forme de valeurs hexadécimales et il n'est bien sûr jamais utilisé directement par les développeurs X. La Xlib (`libX11`) est l'interface de ce protocole avec l'API de programmation X mais elle ne permet d'effectuer que des opérations élémentaires, telles que le tracé de polygone ou l'affichage d'une chaîne de caractères. La bibliothèque Xt (`libXt`, appelée aussi Intrinsics) définit des classes de base d'objets graphiques. Ces classes de bases sont ensuite utilisées par les « toolkits » comme Xaw (`libXaw`, Athena Widgets, du nom du projet Athena associé au développement de X) ou Motif, développé par l'OSF (Open Software Foundation, *http://www.opengroup.org*). À vocation commerciale initialement, ce toolkit est désormais diffusé en Open Source sur *http://www.openmotif.org*.

Le système X a cependant un défaut notoire ; en effet, de par sa grande flexibilité et son indépendance par rapport au matériel, il est relativement coûteux en ressources matérielles et a longtemps été réservé aux stations graphiques haut de gamme. Notez cependant que X est également un logiciel Open Source quoique non-GPL. La licence de X est d'ailleurs moins restrictive que la GPL. Des informations générales sont disponibles sur le site officiel de X Window System, soit *http://www.x.org*. Le site de Kenton Lee (*http://www.rahul.net/kenton*), célèbre consultant spécialiste de X, est également une mine de renseignements et d'articles.

En raison des contraintes induites par le transport de requêtes graphiques à travers le réseau, le fait est que bon nombre de systèmes X sont aujourd'hui constitués d'un serveur et de clients situés sur *la même* machine physique. La complexité amenée par la gestion du système distant est donc peu utilisée bien que celle-ci pénalise les performances.

Un portage de X pour les architectures Unix/x86 est depuis longtemps disponible *via* le projet XFree86 (*http://www.xfree86.org*). La complexité d'une architecture de type x86 est principalement due à la multitude de cartes graphiques disponibles, contrairement aux anciennes stations de travail propriétaires pour lesquelles la carte graphique était fournie par le constructeur de la station.

Par rapport au sujet qui nous occupe dans cet ouvrage (Linux embarqué), nous voyons bien qu'il existe une réelle contradiction dans l'utilisation d'une couche graphique pour ce type de système : la couche graphique de prédilection est par nature lourde, et donc difficile à intégrer dans un environnement réduit *mais* c'est cependant celle pour laquelle nous aurons le plus de choix concernant les logiciels disponibles. Pour ne citer qu'un exemple, bon nombre de *plug-ins* permettant de gérer des formats propriétaires mais très répandus, comme le PDF (Portable Document Format) ou le Flash Macromedia, sont le plus souvent disponibles pour le système graphique X.

Réduction du système X

Un système graphique comme X est souvent en marge de la configuration d'un système embarqué, étant considéré comme « externe » au système d'exploitation lui-même. C'est en partie vrai car la structure de Linux fait qu'il n'y a pas d'imbrication directe entre les couches graphiques et les couches système indispensables. L'avantage évident en est, bien entendu, que l'on peut avoir un système Linux fonctionnel sans aucune autre interface que le mode texte, et donc de très faible taille. La conséquence directe en est toutefois le relativement faible intérêt que les éditeurs principaux de distributions Linux embarqués portent à la partie X. Nous verrons dans la suite du chapitre que la situation évolue avec l'apparition de nouveaux systèmes graphiques (comme Qt) non basés directement sur la technologie X et donc souvent mieux adaptés à un environnement réduit. Le fait est que la réduction de X est complexe et surtout extrêmement liée à l'environnement de la cible. Des ouvrages entiers sont consacrés à X et XFree86, et il est vrai que nous sommes à la limite de la technologie Linux embarquée à part entière (les problèmes seraient les mêmes sur d'autres systèmes comme FreeBSD).

Dans cette section, nous nous efforcerons donc de rester assez généraux même si une solution concrète et fonctionnant dans la majorité des cas sera finalement présentée.

Remarque

Au moment de la rédaction de cet ouvrage, deux versions de XFree86 (la 3 et la 4) cohabitent encore sur certaines distributions, même si XFree86-4 prend largement le pas sur l'autre version de par sa structure plus modulaire au niveau des drivers. Les exemples que nous présentons sont exclusivement basés sur XFree86-4.

Les composants installés par les paquetages XFree86 d'une distribution classique Red Hat 7.2 occupent un espace assez volumineux, d'environ une centaine de Mo. Dans le cas de la distribution Red Hat 7.2, tous les composants XFree86 sont installés par des paquetages RPM du type `XFree86-nom_package.i386.rpm`.

```
# rpm -qa | grep XFree86
XFree86-ISO8859-15-100dpi-fonts-4.1.0-3
XFree86-libs-4.1.0-3
XFree86-xfs-4.1.0-3
XFree86-75dpi-fonts-4.1.0-3
XFree86-ISO8859-15-75dpi-fonts-4.1.0-3
XFree86-twm-4.1.0-3
XFree86-xdm-4.1.0-3
XFree86-4.1.0-3

# du -s /usr/X11R6
102828  /usr/X11R6
```

Il est évident que, dans le cas d'un système embarqué, nous installerons uniquement les composants indispensables à notre application. Si nous détaillons un peu la structure de XFree86, il apparaît que le système est composé des éléments suivants :

- le serveur X
- les clients X
- les bibliothèques X
- les polices de caractères
- divers autres fichiers de configuration dont le fichier principal de configuration du serveur qui est souvent **`/etc/X11/XF86Config`** ou **`/etc/X11/XF86Config-4`**.

Concernant la méthodologie de réduction du système X, le principe appliqué sera donc à peu près le même que s'agissant de la réduction du système Linux lui-même.

Le serveur X est quant à lui indispensable mais on peut réduire son empreinte mémoire en sélectionnant uniquement les pilotes graphiques adaptés au matériel et les « extensions » indispensables à notre application (dans une installation Linux classique, plusieurs pilotes peuvent être installés par défaut). Si nous comparons une version complète à une version réduite, nous pouvons constater que le rapport est d'environ 20.

```
# du -s /usr/X11R6.gen/lib/modules
1156    /usr/X11R6.gen/lib/modules
# du -s /usr/X11R6.orig/lib/modules
21004   /usr/X11R6.orig/lib/modules
```

Ce type d'optimisation, très spectaculaire, peut être obtenue en utilisant la carte graphique en mode VESA (*Video Electronics Standards Association*, *http://www.vesa.org*). L'association VESA a publié un standard de BIOS graphique (VBE pour *VESA BIOS Extension*) permettant d'utiliser toutes les cartes graphiques compatibles VBE avec un jeu d'instructions commun. La version courante est la 3.0, le standard étant disponible en téléchargement sur *http://www.vesa.org/vbelink.html*. XFree86 nécessite des cartes compatibles VBE 2.0 pour fonctionner, ce qui est le cas de la quasi-totalité des *chipsets* graphiques du commerce. Le mode VESA est disponible en installant un minimum de modules pour XFree86.

```
# pwd
/usr/X11R6/lib/modules
```

```
# ls -l
total 756
drwxrwxr-x    2 root     root         4096 mai  6 12:40 drivers
drwxrwxr-x    2 root     root         4096 avr 19 13:08 extensions
drwxrwxr-x    2 root     root         4096 avr 19 12:59 fonts
drwxrwxr-x    2 root     root         4096 avr 19 11:17 input
-rwxr-xr-x    1 root     root        27286 avr 19 12:58 libddc.a
-rwxr-xr-x    1 root     root       147402 avr 19 12:58 libfb.a
-rwxr-xr-x    1 root     root       188930 avr 19 12:58 libint10.a
-rwxr-xr-x    1 root     root        62372 avr 19 12:58 libpcidata.a
-rwxr-xr-x    1 root     root        30942 avr 19 12:58 libramdac.a
-rwxr-xr-x    1 root     root       121454 avr 19 12:58 libscanpci.a
-rwxr-xr-x    1 root     root        27262 avr 19 12:58 libshadow.a
-rwxr-xr-x    1 root     root        18494 avr 19 12:58 libshadowfb.a
-rwxr-xr-x    1 root     root        10210 avr 19 12:58 libvbe.a
-rwxr-xr-x    1 root     root        19786 avr 19 12:58 libvgahw.a
drwxrwxr-x    2 root     root         4096 avr 19 11:46 linux
-r--r--r--    1 root     root        26253 jun  6  2001 v10002d.uc
-r--r--r--    1 root     root        35007 jun  6  2001 v20002d.uc
```

La partie driver elle-même étant réduite à :

```
# ls -l drivers/
total 40
-rwxr-xr-x    1 root     root        21134 jun  6  2001 vesa_drv.o
-rwxr-xr-x    1 root     root        14809 jun  6  2001 vga_drv.o
```

De même, concernant le fichier **`XF86Config`**, on sélectionnera le pilote en configurant la section **`Devices`** comme suit :

```
Section "Device"
    Identifier  "MyVidCard"
    Driver      "vesa"
    VideoRam    2048
    # Insert Clocks lines here if appropriate
EndSection
```

Le mode VESA est cependant moins performant que les modes accélérés des cartes récentes. En cas de besoin réel de hautes performances, il conviendra d'effectuer la configuration du serveur X en fonction du mode accéléré.

Selon la sélection des clients et des applications X locales au système, on peut en déduire les bibliothèques nécessaires et donc celles que l'on peut supprimer. Pour cela, on peut utiliser le script **`mklibs.sh`** décrit au chapitre 5. Comme on a pu le préciser, ce script ne résoudra cependant pas les problèmes des bibliothèques chargées lors de l'exécution. La seule solution dans ce cas consiste à travailler de manière empirique en effectuant des modifications du fichier **`XF86Config`**, en ajoutant ou supprimant des modules au fur et à mesure et en validant les modifications par un test du type :

```
xinit -- -depth 16
```

En fonction des mêmes critères, on peut sélectionner les polices de caractères à conserver.

Par défaut, XFree86-4 utilise un programme « serveur » de fontes appelé **xfs** (X Font Server). Cela se traduit par la ligne de configuration suivante :

```
    FontPath   "unix/:7100"
```

la configuration réelle étant déportée dans le fichier **/etc/X11/fs/config** du programme **xfs**. Dans le cas d'une version réduite, nous n'utiliserons pas **xfs** et les polices de caractères seront directement déclarées dans le fichier **XF86Config**.

```
    FontPath   "/usr/X11R6/lib/X11/fonts/misc/"
    FontPath   "/usr/X11R6/lib/X11/fonts/75dpi/:unscaled"
    FontPath   "/usr/X11R6/lib/X11/fonts/75dpi/"
```

Le nombre de polices doit donc être précisément défini en fonction des besoins de l'application mais, en toute rigueur et indépendamment des considérations d'esthétique, un serveur X a besoin d'une seule police pour fonctionner (la police est définie comme **fixed**). Au total, l'empreinte mémoire nécessaire pour un système X réduit est de l'ordre de 8 Mo, sachant que la liste des clients disponibles est réduite au strict minimum.

```
# cd /usr/X11R6/bin
# ls -l
total 3064
-rwxr-xr-x    1 root     root         2166 avr 23 16:08 startx
-rwxr-xr-x    1 root     root       153915 avr 19 12:55 twm
lrwxrwxrwx    1 root     root            7 mai  6 11:42 X -> XFree86
-rwxr-xr-x    1 root     root        35736 avr 19 12:55 xauth
-rws--x--x    1 root     root      1746388 avr 19 12:56 XFree86
-rwxr-xr-x    1 root     root        14155 avr 19 12:55 xinit
-rwxr-xr-x    1 root     root        11919 avr 19 12:55 xkill
-rwxr-xr-x    1 root     root        37427 jun  6  2001 xmodmap
-rwxr-xr-x    1 root     root        32494 jun  6  2001 xrdb
-rws--x--x    1 root     root       177496 jan  8  2000 xterm
-rwxr-xr-x    1 root     root       891516 avr 22 13:43 Xvesa

# du -c -s /usr/X11R6 /etc/X11
7844    /usr/X11R6.gen
632     /etc/X11.gen
8476    total
```

Un serveur X minimal (*Xkdrive*)

Indépendamment de l'empreinte mémoire sur le disque, X est relativement gourmand en mémoire vive. Les sources de XFree86-4 incluent cependant un serveur très optimisé tant au niveau de l'empreinte mémoire que de la RAM utilisée à l'exécution. Le serveur **Xkdrive** contient quelques limitations, par exemple l'absence de support des polices vectorielles. Autre limitation : il ne fonctionne que sous Linux/x86 et pour un nombre limité de *chipsets* graphique. Il fonctionne cependant très bien en mode VESA ou bien en utilisant le *frame-buffer* Linux décrit à la section suivante. La page de manuel associée est disponible en ligne sur *http://www.xfree86.org/current/Xkdrive.1.html*.

Contrairement aux autres composants, ce serveur **Xkdrive** n'est en général pas disponible sur les paquetages RPM binaires des distributions classiques et la meilleure solution est de partir directement des sources de XFree86-4. Pour cela, il faut choisir un bon miroir, comme *ftp://ftp.free.fr*.

```
# ncftp ftp.free.fr
NcFTP 3.0.0 beta 21 (October 04, 1999) by Mike Gleason (ncftp@ncftp.com).
Connecting to 212.27.32.12...
ProFTPD 1.2.1 Server (ProFTPD) [ftpmirror.proxad.net]
Logging in...
Anonymous access granted, restrictions apply.
Logged in to ftp.free.fr.
ncftp / > cd mirrors/ftp.xfree86.org/XFree86/4.2.0/source
ncftp ...g/XFree86/4.2.0/source > dir
-rw-r--r--   1 701      1190      1411717   jan 18 23:43   doctools-1.3.tgz
-rw-rw-r--   1 701      1190      1258628   jan 28 16:27   FILES
-rw-rw-r--   1 701      1190          331   jan 28 16:27   SUMS.md5
-rw-rw-r--   1 701      1190          289   jan 28 16:27   SUMS.md5sum
-rw-r--r--   2 701      1190      1036536   jan 26 18:03   utils-1.0.1.tgz
-rw-r--r--   2 701      1190      1036536   jan 26 18:03   utils.tgz
-rw-r--r--   1 701      1190     25961532   jan 18 23:43   X420src-1.tgz
-rw-r--r--   1 701      1190     23227328   jan 18 23:43   X420src-2.tgz
-rw-r--r--   1 701      1190      9306679   jan 18 23:43   X420src-3.tgz
```

Les sources sont divisées en trois archives **X420src-X.tgz** qu'il faut rapatrier. Après extraction des trois archives, on obtient le répertoire xc contenant entre autres le fichier de documentation **INSTALL-X.org**. Le but n'étant pas de construire entièrement XFree86-4 mais seulement le serveur Xkdrive, il faut créer un fichier **host.def** sur le répertoire **xc/config/cf**. Ce fichier a l'allure suivante :

```
#define BuildServersOnly YES
#define KDriveXServer YES
#define TinyXServer YES
#define XfbdevServer NO
#define XvesaServer YES
#define BuildType1 YES
```

On lance ensuite la compilation en exécutant :

```
make World >& world.log
```

et on peut surveiller la compilation en lançant :

```
tail -f world.log
```

Lorsque la compilation est terminée avec succès, on doit avoir à la fin de **world.log** :

```
Full build of Release 6.6 of the X Window System complete.
```

Le fichier **Xvesa** doit également être construit.

```
$ ls -l programs/Xserver/Xvesa
-rwxrwxr-x    1 pierre   pierre     838540 mai  3 14:31 programs/Xserver/Xvesa
```

On note la taille relativement modeste de cet exécutable (« seulement » 800 Ko) par rapport à la version standard. On peut tester le serveur en exécutant :

```
# Xvesa -screen 1024x768x16 &
```

qui utilisera une résolution de 1024 sur 768 pixels en mode 16 bits (65 536 couleurs).

frame-buffer (console graphique)

Nous avons vu pour l'instant que la seule solution pour utiliser Linux en mode graphique était de passer par X Window System. Même si X peut être réduit pour convenir à un environnement embarqué, il ne représente pas forcément la meilleure solution.

La console graphique ou *frame-buffer* permet de piloter les modes graphiques haute résolution directement depuis le noyau Linux, ce qui n'est pas le cas de la majorité des serveurs X qui le font en mode utilisateur. Une des applications principales du frame-buffer est la possibilité d'afficher un logo de pingouin au démarrage de LINUX mais nous allons voir que ce n'est pas la seule.

Configuration du frame-buffer

Le support frame-buffer doit être activé *via* la procédure standard de configuration du noyau Linux en utilisant le menu *Console drivers/Frame-buffer*. On obtient alors l'écran décrit ci-après :

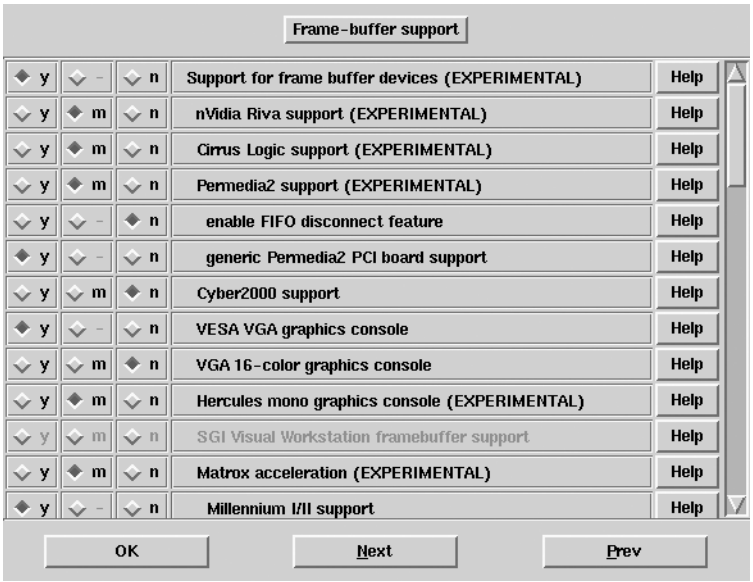


Figure 12-3

Configuration du frame-buffer.

Comme dans le cas de X, le mode VESA est disponible à condition que le chipset graphique soit compatible VBE 2.0 ou supérieur. Certaines cartes comme les MATROX ou les ATI disposent également de pilotes spécifiques bien qu'elles fonctionnent également en mode VESA. La description des différents modes graphiques accessibles en mode VESA est disponible dans le fichier `Documentation/fb/vesafb.txt` des sources du noyau 2.4. Si l'on veut valider le support d'une carte graphique particulière (non-VESA), il faudra cocher l'option correspondante.

Lorsque le noyau est compilé et installé, on peut utiliser LILO pour indiquer le mode graphique à utiliser au démarrage.

Par défaut, LILO utilise la ligne suivante pour démarrer le système en mode texte :

```
vga = NORMAL
```

Le document `vesafb.txt` donne la liste des codes à utiliser en fonction des résolutions graphiques et des nombres de couleurs, de 256 à 16 millions.

Tableau 12-1. Codes VESA pour frame-buffer.

	640 × 480	800 × 600	1024 × 768	1280 × 1024
256	0x301	0x303	0x305	0x307
32 K	0x310	0x313	0x316	0x319
64 K	0x311	0x314	0x317	0x31A
16 M	0x312	0x315	0x318	0x31B

Si le fichier `/etc/lilo.conf` est modifié en ajoutant la ligne :

```
vga = ASK
```

l'utilisateur devra saisir la valeur hexadécimale du mode graphique lors du démarrage du système, soit par exemple 317 pour une résolution de 1024 × 768 en 64 K couleurs (16 bits). Si l'on désire utiliser un mode graphique en permanence, on pourra utiliser une section LILO du style :

```
image=/boot/bzImage-2.4.18_fb_vesa
# 1024x768x16bpp
       vga=791
# 640x480x16bpp
#        vga=785
# 800x600x16bpp
#       vga=788

        label=linux18fbv
        read-only
        root=/dev/hda4
```

Dans ce cas, la valeur doit être convertie en *décimal* car LILO ne gère pas les valeurs hexadécimales. Si tout se passe bien, le système doit basculer en mode graphique et

afficher le logo du petit pingouin. On peut également configurer le mode graphique en passant l'option **video=** à LILO lors du démarrage du système soit par exemple :

```
LILO: linux18fbv video=vesa:1024x768-16
```

pour le même mode graphique $1024 \times 768 \times 16$ bpp. La ligne pourra bien entendu être ajoutée systématiquement avec une directive **APPEND** dans le fichier **/etc/lilo.conf**. Dans le cas d'une carte ATI Mach64 en mode non VESA, on utiliserait :

```
LILO: linux18fbv video=atyfb:1024x768-16
```

Manipulation du frame-buffer

Lorsque le frame-buffer est fonctionnel, il est disponible *via* le device **/dev/fb0**. La notion de fichier banalisé permet d'effectuer une copie d'écran en opérant une simple copie du device vers un autre fichier.

```
cp /dev/fb0 /tmp/test.raw
```

On obtient alors un fichier en format binaire correspondant au contenu de la mémoire du frame-buffer. On peut le convertir dans un format plus sympathique en utilisant le petit programme **raw2ppm** qui suit :

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
  int len;
  unsigned short buf[256];
  FILE *f;
  int w, h;

  w = 640;
  h = 480;

  printf("P6\n%d %d\n255\n", w, h);
  f = fopen(argv[1], "rb");
  while ((len = fread(buf, 2, 256, f)) != 0) {
    int i;

    for (i = 0; i < len; i++) {
      unsigned char r, g, b;
       #ifdef BYTESWAP
       /* Swap the byte order */
       buf[i] = ((buf[i] & 0xff00) >> 8) + ((buf[i] & 0x00ff) << 8);
       #endif
       r = (buf[i] & 0xF800) >> 8;
       g = (buf[i] & 0x07E0) >> 3;
       b = (buf[i] & 0x001F) << 3;
      printf("%c%c%c", r, g, b);
    }
  }
}
```

On en déduit l'enchaînement de commandes qui conduit à un fichier PNG facilement manipulable.

```
raw2ppm /tmp/test.raw |  pnmtopng > /tmp/test.png
```

Les toolkits graphiques

Dans la partie consacrée à X, nous avons cité les toolkits ATHENA et MOTIF largement utilisés dans les applications habituelles. Les critères d'une application embarquée sont cependant très différents, et nous voyons apparaître de plus en plus d'outils dédiés à un environnement embarqué, particulièrement sous Linux. En laissant de côté MOTIF et ATHENA, nous allons ici décrire brièvement le toolkit commercial Qt/Embedded édité par Trolltech et le projet MicroWindows/NanoX.

Qt/Embedded

La société norvégienne Trolltech (*http://www.trolltech.com*) développe depuis plusieurs années le toolkit Qt. Initialement développé pour Unix et l'environnement X11, Qt est un toolkit multi-plate-forme qui permet de réaliser des applications graphiques portables entre des environnements Unix, Windows et MacOS. Qt est à la base du bureau KDE (*http://www.kde.org*), très utilisé sous Linux. Qt est aussi le « concurrent » direct de GTK (*http://www.gtk.org*), à la base du bureau GNOME (*http://www.gnome.org*). Notez qu'il existe une version de GTK tournant sur le frame-buffer Linux (voir *http://developer.gnome.org/doc/API/2.0/gtk/gtk-framebuffer.html*).

La société Trolltech a récemment étendu son développement en réalisant une version de Qt appelée *Qt/Embedded*, adaptée à un environnement Linux embarqué et pouvant utiliser le frame-buffer décrit précédemment. Qt et Qt/Embedded sont des produits commerciaux mais il existe des versions GPL strictement identiques aux distributions commerciales, mis à part le fait qu'elles n'existent que pour l'environnement Unix et pas pour Windows ni MacOS. Notez que la licence utilisée est la GPL et non pas la LGPL (*Lesser GPL*) ce qui limite l'utilisation des versions GPL de Qt et Qt/Embedded à des applications elles-mêmes sous GPL. La version GPL de QT/Embedded est disponible à l'adresse *http://www.trolltech.com/download/qtembedded.html.*

Cette page rappelle clairement les termes de la licence d'utilisation de la version libre de Qt/Embedded. Après extraction de l'archive, on obtient le répertoire **`qt-embedded-free-3.3.3`**. Pour préparer la compilation, on doit se positionner sur le répertoire et valider les variables d'environnement **`QTDIR`** et **`LD_LIBRARY_PATH`**.

```
cd  qt-embedded-free-3.3.3
export QTDIR=`pwd`
export LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
```

On doit ensuite configurer l'environnement en utilisant le script **configure** auquel on peut passer différentes options. La liste des options est disponible en exécutant :

```
./configure --help
```

Attention

Pour générer correctement la bibliothèque et les exemples, il est indispensable de spécifier la liste des « profondeurs » de frame-buffer utilisables (8, 16, 24 ou 32 plans).

Pour ce faire, on utilisera la ligne :

```
./configure -depths 16,24,32
```

sachant que Trolltech annonce que le support 8 plans est expérimental. Après avoir répondu aux quelques questions en validant les réponses par défaut, on peut enfin taper **make** pour générer la bibliothèque et les exemples. Notez que Qt/Embedded est entièrement écrit en C++, ce qui entraîne un temps de compilation assez long comparé au C.

Lorsque la compilation est terminée, on peut aller dans le répertoire des exemples et lancer le démonstrateur.

```
# cd examples/launcher
# ./start_demo
```

L'écran affiché a l'allure suivante :

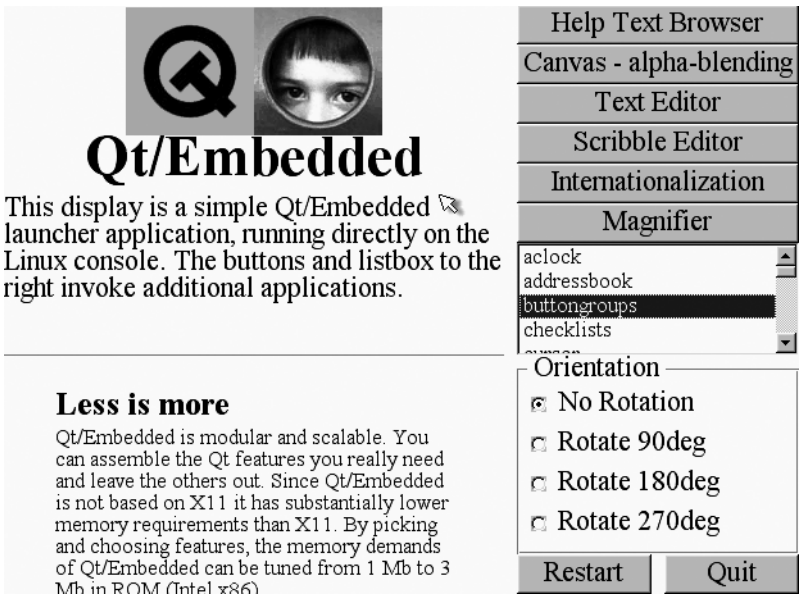


Figure 12-4

Démonstrateur Qt/Embedded.

Si l'on clique sur le mot *widgets* à la fin de la liste déroulante de droite, on obtient l'écran suivant qui donne un aperçu des différents objets graphiques disponibles sous Qt/Embedded. On remarquera que ces objets sont identiques à ceux de Qt.

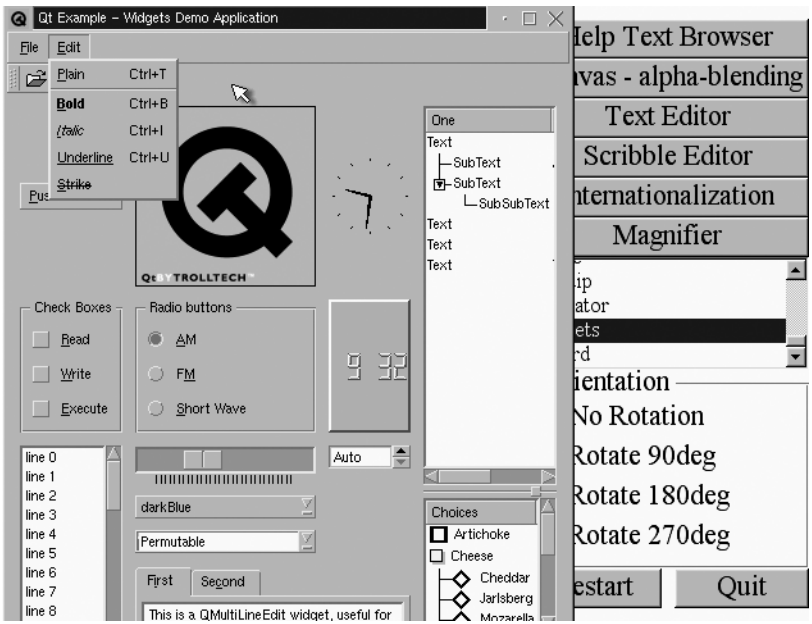


Figure 12-5
Widgets Qt/Embedded.

GTK-Embedded

Cette bibliothèque a pour origine l'outil d'édition graphique GIMP. À l'époque elle fut créée comme alternative libre (et plus performante) à la bibliothèque OSF-Motif qui était alors propriétaire. Depuis, elle a servi de base à l'environnement GNOME. Cette bibliothèque est écrite en langage C et elle est diffusée sous licence LGPL, ce qui permet de l'utiliser pour ses propres développements sans obligatoirement placer son propre code sous GPL. Au niveau des fonctionnalités, elle est relativement proche de Qt et sera donc réservée aux applications graphiques relativement complexes.

La compilation de GTK+ est plus complexe que celle de Qt car elle nécessite la présence de plusieurs bibliothèques annexes pour fonctionner (soit Glib, Pango et ATK). Ces bibliothèques sont disponibles en téléchargement sur le site de GTK+ (*http://www.gtk.org*). Pour ce test, nous avons utilisé les versions suivantes des bibliothèques :

- atk-1.8.0
- glib-2.4.8

- gtk+-2.4.14
- pango-1.4.1

Il faut tout d'abord compiler la bibliothèque glib en utilisant la procédure classique, soit :

```
$ ./configure -prefix=/usr
$ make
# make install
```

On peut ensuite faire de même pour les bibliothèques Pango et ATK.

Pour la bibliothèque GTK+, il faut indiquer que l'on va utiliser le frame-buffer et non la sortie X11 qui est la configuration par défaut. Avant cela, il faut appliquer un petit patch à la bibliothèque car il semble y avoir un problème de fonction non définie si l'on utilise la sortie frame-buffer. Il est inspiré d'un patch diffusé pour GTK+-2.4.0 par Frédéric Crozat (développeru de Mandriva) mais il semblerait que celui-ci n'ait pas été pris en compte par les développeurs de GTK+. Le patch est disponible dans l'espace de téléchargement de l'ouvrage (*www.editions-eyrolles.com*).

La procédure de compilation est donc la suivante, tout d'abord l'application du patch à partir du répertoire des sources. :

```
$ patch -p0 < /tmp/gtk_2.4.4_linux-fb.patch
patching file gdk/linux-fb/gdkdrawable-fb2.c
patching file gdk/linux-fb/gdkfont-fb.c
patching file gdk/linux-fb/gdkwindow-fb.c
```

Ensuite viennent la génération de l'environnement via **`configure`** en précisant la sortie frame-buffer, puis la compilation et enfin l'installation :

```
$ ./configure --prefix=/usr --with-gdktarget=linux-fb
$ make
# make install
```

Lorsque tout est installé on peut utiliser le programme de démonstration présent dans le répertoire **`demos/gtk-demo`** des sources de GTK+. Ce programme doit bien entendu être appelé depuis la console frame-buffer Linux en ayant pris soin de stopper le service **`gpm`** (qui gère la souris en mode texte) si ce dernier était actif.

```
# cd demos/gtk-demo
# ./gtk-demo
```

L'exécution du programme de test provoque l'affichage de la fenêtre présentée sur la figure page suivante.

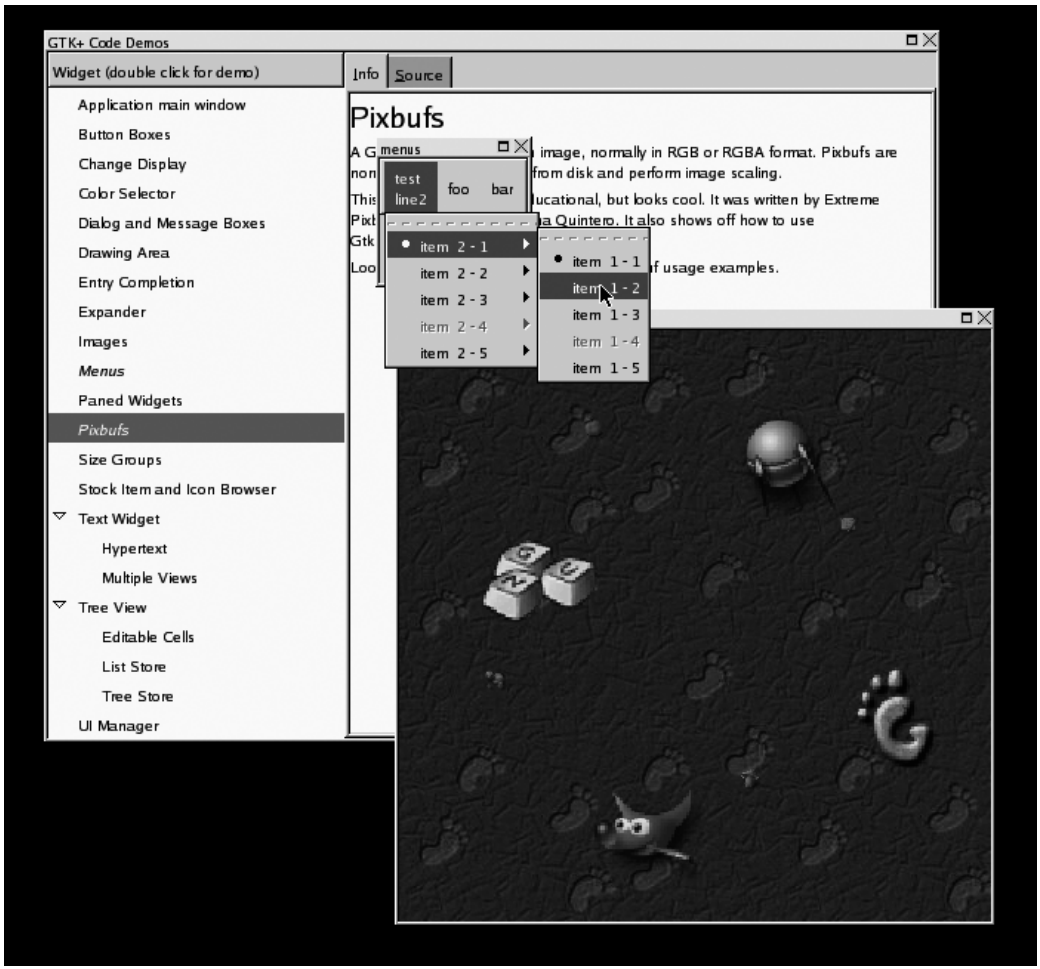


Figure 12-6
Démonstration de GTK-Embedded.

Microwindows et Nano-X

À la différence de Qt/Embedded, MicroWindows et Nano-X sont des produits non commerciaux. La page d'accueil du projet est située à l'adresse *http://www.microwindows.org*. Ces bibliothèques sont diffusées sous MPL (*Mozilla Public licence*, *http://www.mozilla.org/MPL*), ce qui signifie que l'on peut baser des applications commerciales sur MicroWindows et Nano-X à condition que les modifications éventuelles sur les bibliothèques restent en Open Source. La bibliothèque MicroWindows utilise une API similaire à celle de Windows alors que Nano-X utilise un mécanisme client-serveur proche du fonctionnement

de la Xlib sous X11. Même si ces bibliothèques n'ont pas le niveau de fonctionnalité de Qt/Embedded ou GTK-Embedded, leur mise en place est très légère et peut bien convenir à des applications graphiques simples. Notez qu'il existe une version Nano-X du navigateur Mozilla-0.9.4.

Sous Linux, les bibliothèques peuvent être utilisées sous X11, ou bien recourent au frame-buffer. L'archive est disponible à partir de la page d'accueil du projet et, une fois extraite, on obtient le répertoire **`microwindows-0.89pre8`** (selon la version).

La configuration des bibliothèques est située dans le fichier **`src/config`** et les valeurs par défaut génèrent une version frame-buffer. Si l'on souhaite générer la version X11, il faut modifier la ligne suivante :

```
# X Window screen, mouse and kbd drivers
X11                     = Y
```

Le support du frame-buffer est exclusif avec le support X11. Il est activé au moyen des lignes suivantes :

```
FRAMEBUFFER             = Y
FBVGA                   = Y
```

Dans tous les cas, la compilation s'effectue par une simple commande **`make`**. Les programmes de démonstration sont ensuite disponibles sur le répertoire **`src/bin`**. Dans le cas de l'utilisation de X11, le support de la souris est assuré par X. Dans le cas de l'utilisation du frame-buffer, la solution la plus simple est d'utiliser le programme **`gpm`**, ce qui est d'ailleurs le comportement par défaut dans le fichier de configuration.

```
GPMMOUSE                = Y
```

Attention

Pour l'utilisation de la souris avec gpm, il est nécessaire de lancer préalablement gpm avec la commande `gpm -R -t ps2`.

Pour tester le fonctionnement de Nano-X, on doit tout d'abord lancer le serveur, puis les clients. Dans l'exemple qui suit, nous utiliserons les clients **`nanowm`** (gestionnaire de fenêtre ou *window-manager*), **`nxterm`** (émulateur de terminal), **`nxclock`** (pendule) et **`ntetris`** (jeu TETRIS).

```
# cd src/bin
# ./nano-X &
# ./nanowm &
# ./nxterm &
# ./nxclock &
# ./ntetris &
```

Le résultat est indiqué dans la figure ci-après.

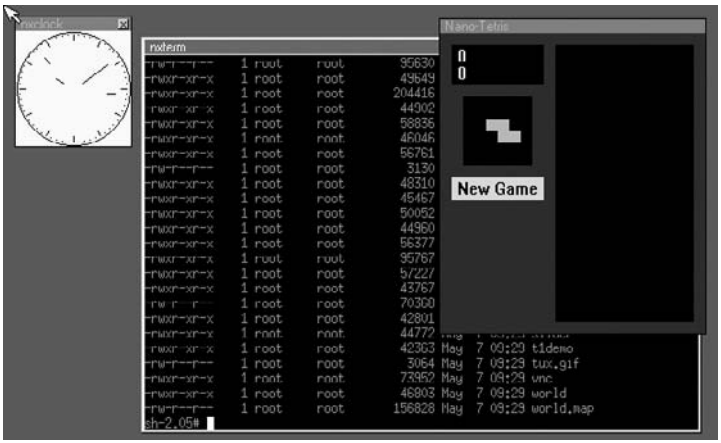


Figure 12-7
Démonstration de Nano-X.

Plusieurs démonstrations de MicroWindows sont disponibles sur le même répertoire. Si l'on fait :

```
# ./mdemo
```

on obtient l'écran présenté ci-après :

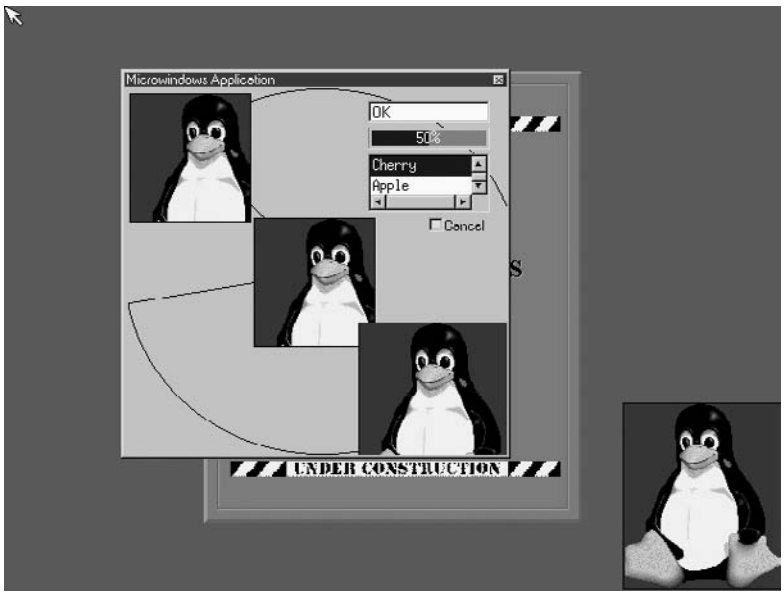


Figure 12-8
Démonstration de MicroWindows.

Une bibliothèque d'affichage LCD: LCDproc

Même si les systèmes embarqués deviennent de plus en plus puissants, les applications industrielles n'ont pas toujours besoin d'un affichage complexe. Bon nombre de systèmes se contentent d'un afficheur à cristaux liquides type LCD. Pour ce faire, le projet LCDproc (*http://lcdproc.omnipotent.net*) fournit une bibliothèque puissante, simple, et utilisable sur plusieurs afficheurs du commerce. En plus de cela, la bibliothèque fournit un pilote de test utilisant l'interface *curses* ce qui permet de mettre au point l'application sans disposer d'afficheur réel.

Le principe de LCDproc est basé sur une technique de client/serveur. Le serveur est chargé d'effectuer l'affichage sur l'écran LCD et reçoit pour cela les requêtes des clients à travers un socket TCP sur un le port 13666. Par défaut le serveur et les clients sont exécutés sur une même machine (soit localhost) mais rien n'empêcherait d'effectuer l'affichage à travers Internet.

La compilation de LCDproc ne présente pas de difficultés particulières. Après extraction de l'archive, le mieux est de compiler le paquetage en incluant tous les pilotes d'écran disponibles :

```
$ ./configure --enable-drivers=all
$ make
```

Pour tester le système, il faut tout d'abord sélectionner un pilote dans le fichier de configuration du serveur soit `LCDd.conf`. Dans notre cas, nous utiliserons le pilote de test curses.

```
[server]
# Server section with all kinds of settings for the LCDd server

#Driver=none
Driver=curses
#Driver=HD44780
```

Il faut ensuite exécuter le programme serveur dans un terminal comme suit. L'option `-s` indique que l'affichage des messages d'erreurs utilisera le démon syslog.

```
$ ./server/LCDd -c LCDd.conf -s
```

La fenêtre prend alors l'allure suivante :

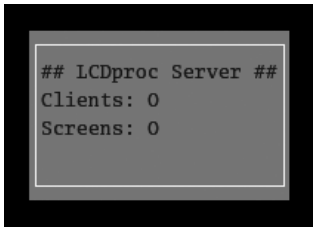


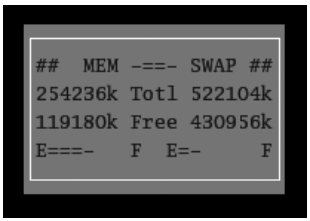
Figure 12-9
Démarrage du serveur LCDproc.

On peut alors utiliser le client **`lcdproc`** fourni pour effectuer un test d'affichage. Ce client affiche l'état du système en temps réel.

```
$ ./clients/lcdproc/lcdproc
```

Ce qui provoque l'affichage suivant :

Figure 12-10
Serveur LCDproc et client lcdproc.



Il est bien entendu possible de piloter directement le serveur depuis un simple client **`telnet`** puisque nous utilisons le protocole TCP. La description des différents objets disponibles est présente dans le fichier **`docs/netstuff.txt`** de la distribution LCDproc.

Dans notre exemple nous effectuons une session telnet dans laquelle :

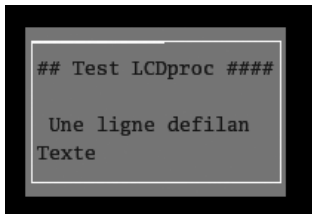
- Nous initialisons le protocole LCDproc.
- Nous créons un écran d'affichage.
- Nous créons trois objets *title*, *string* et *scroller* auxquels nous affectons des valeurs.

Le texte en gras représente les commandes tapées par l'utilisateur, le reste du texte représente les réponses du serveur :

```
$ telnet localhost 13666
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello
connect LCDproc 0.4.5 protocol 0.3 lcd wid 20 hgt 4 cellwid 5 cellhgt 8
screen_add s
success
listen s
widget_add s t title
success
widget_set s t "Test LCDproc"
success
widget_add s str string
success
widget_set s str 1 4 Texte
success
widget_add s scroll scroller
success
widget_set s scroll 2 3 18 3 h 1 "Une ligne defilante"
success
```

La fenêtre du serveur LCDproc a alors l'allure suivante :

Figure 12-11
Serveur LCDproc et client telnet.



Navigateurs et serveurs web

Le navigateur web ou *browser* est de plus en plus utilisé dans les applications industrielles car il permet d'accéder à des équipements de manière simple, conviviale, et indépendamment de l'architecture. Nombre de systèmes embarqués sont donc équipés de serveur HTTP comme le programme Apache (*http://www.apache.org*) ou d'autres programmes serveur HTTP plus réduits en cas de système à faible empreinte mémoire. Parmi ces programmes, on peut citer Boa (*http://www.boa.org*) et MHTTPD (*http://www.muquit.com/muquit/software/mhttpd/mhttpd.html*). Ces deux serveurs supportent les scripts CGI mais sont destinés à recevoir un faible nombre de requêtes par comparaison avec un programme comme Apache. L'utilisation pour une configuration à distance sera parfaitement adaptée.

Dans le cas de systèmes équipés d'interface graphique, il peut être intéressant d'utiliser un navigateur, soit parce que le système est destiné à ce type de tâche (*set-top box* ou terminal léger) ou parce qu'il sera plus simple de concevoir une interface basée sur des pages HTML, du JavaScript et des CGI que de développer une interface dédiée. Dans ce cas, la même interface sera disponible en mode local ou bien en mode distant, ce qui est très intéressant à bien des titres. Malheureusement, il faut admettre qu'un navigateur web « classique » est un composant extrêmement complexe, devant intégrer un grand nombre d'extensions utilisées par les sites web actuels. Parmi ces extensions nous pouvons citer les *frames* (multi-fenêtres), JavaScript, Flash, Java et les *plug-ins* divers et variés. Le support de ces extensions conduit à une surenchère dans la complexité des navigateurs comme Internet Explorer 6 (IE 6) ou Nescape Communicator 6 (NS 6). Cette complexité entraîne bien entendu les mêmes excès dans la consommation de ressources machines ou d'espace disque utilisé, critères une fois de plus peu compatibles avec un environnement embarqué.

Les navigateurs peuvent être séparés en deux catégories. La première limite le navigateur à un outil de configuration du système local ou de systèmes distants à vocation industrielle. Dans ce cas-là, le but est de présenter l'interface la plus simple et la plus efficace possible, et l'utilisation d'extensions lourdes et coûteuses n'est certainement pas la meilleure solution. Le navigateur Lynx utilise une interface texte basée sur la bibliothèque Curses citée au chapitre 5. Il ne permet pas d'utiliser d'images mais supporte cependant l'utilisation de scripts CGI. Il est en outre doté de fonctions intéressantes permettant

d'automatiser certaines procédures qui utilisent le protocole HTTP. La page d'accueil du produit se trouve sur *http://lynx.browser.org*.

Le navigateur Dillo (*http://dillo.cipsga.org.br*) est entièrement écrit en C et occupe environ 200 Ko dans sa version binaire. Même si Dillo n'inclut pas les mêmes fonctions, cette valeur soutient aisément la comparaison avec les dizaines de Mo nécessaires pour un navigateur classique. Dillo utilise le toolkit GTK cité précédemment. Il ne supporte pas pour l'instant les frames mais il est très rapide et peut être très bien adapté dans le cas de pages HTML à fenêtre unique. La figure 12-12 donne un aperçu du rendu graphique de Dillo.

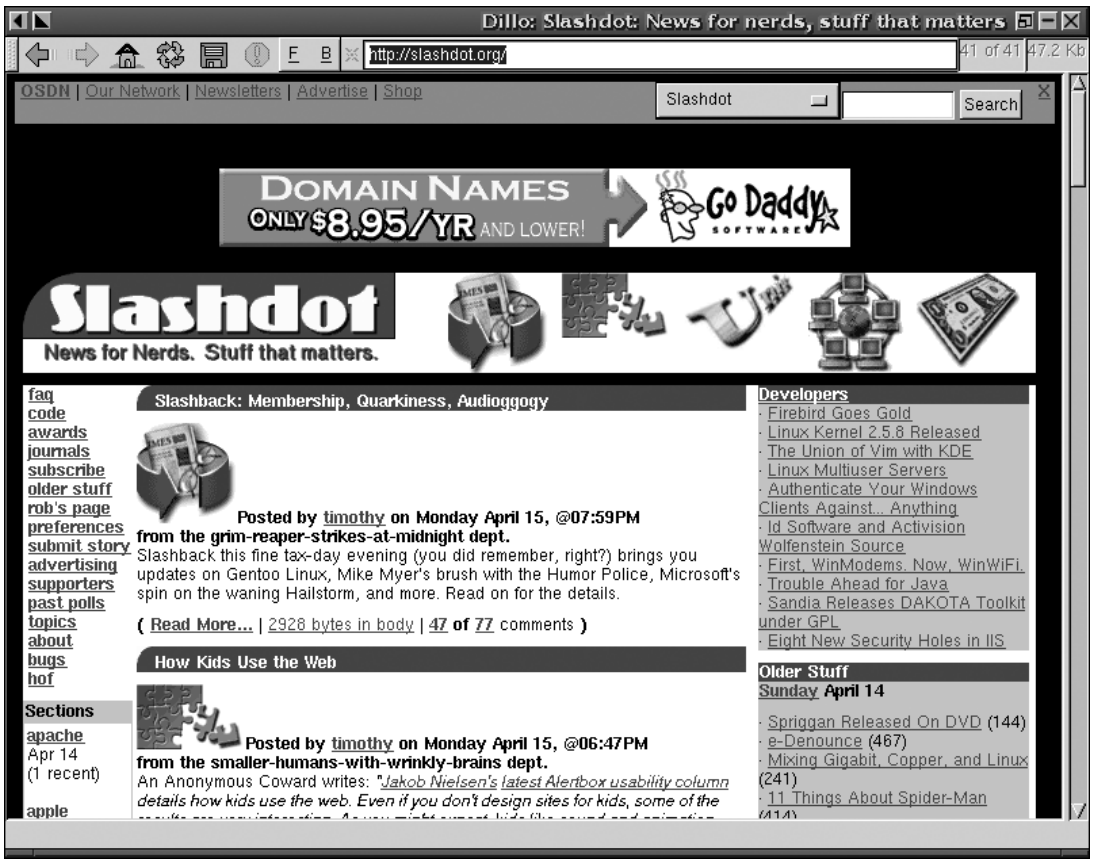


Figure 12-12
Le navigateur Dillo.

Outre ces deux produits, la page *WWW-Browsers for LINUX* située sur *http://www.itp.uni-hannover.de/~kreutzm/en/lin_browser.html* donne une liste impressionnante de navigateurs web utilisables sous Linux.

La seconde catégorie concerne les navigateurs qui doivent s'approcher le plus possible des navigateurs classiques. Pour ces derniers, il est nécessaire que les extensions graphiques multimédias citées précédemment soient disponibles. Ce type de navigateur équipera les systèmes embarqués du genre *set-top boxes*, terminaux légers ou PDA. Le défi est de taille car il n'y a pratiquement aucun standard officiel autour des extensions du format HTML (à peine quelques « avis » du W3C concernant la compatibilité HTML). Le fait est que la loi est faite par ceux qui détiennent le marché du navigateur le plus répandu, en l'occurrence Microsoft avec IE. La logique de développement des concepteurs de pages graphiques est aujourd'hui simple : elle consiste à créer les pages avec des outils de conception dédiés (la plupart du temps sous Windows) puis ensuite à tester exclusivement (ou presque) sous IE, ce dernier devenant donc la référence absolue en manière de compatibilité des pages. La tâche des concurrents du navigateur IE n'en est que rendue plus ardue.

Il est un fait également que, si IE utilise plusieurs dizaines de Mo pour décoder toutes les extensions web, il sera bien difficile à un produit spécialisé « embarqué » de faire exactement la même chose dans un volume environ 10 fois moindre. Dans ce domaine, la compatibilité sera partielle, mais jamais totale, et c'est certainement une des raisons pour lesquelles la diffusion des consoles Internet et *set-top boxes* a toujours été décevante par rapport au PC classique. Il existe cependant quelques éditeurs qui effectuent un travail remarquable sur le sujet.

Le plus célèbre est certainement Opera (*http://www.opera.com*) qui diffuse depuis plusieurs années un navigateur multi-plate-forme (Linux, Windows, MacOS et autres) basé sur la bibliothèque Qt. La version standard d'Opera a la réputation non usurpée d'être extrêmement rapide par rapport à ses concurrents institutionnels. Opera est compatible JavaScript 1.3 et peut également utiliser le *plug-in* Java fourni par Sun Microsystems. La version 6.0 est disponible gratuitement en téléchargement et l'éditeur annonce déjà 1 million de téléchargements de la version Linux en mai 2002. Opera a depuis quelque temps investi dans la technologie embarquée en développant une version dédiée, basée sur Qt/Embedded. Opera annonce le produit comme tournant dans 3 Mo de RAM et occupant 1,5 Mo d'espace disque. Notez que cette version n'inclut pas le support de Java. Pour ce dernier, on devra utiliser une machine virtuelle externe comme le produit Jeode fourni par la société Insignia (*http://www.insignia.com*).

Il est important de noter que le navigateur Opera n'est pas un produit Open Source et que son utilisation à des fins commerciales devra faire l'objet de paiement de redevances à l'éditeur. Opera fournit cependant une documentation très détaillée et certains composants Open Source afin de permettre la personnalisation du navigateur en fonction des besoins de l'application. Opera/Embedded est déjà utilisé dans plusieurs produits industriels comme le PDA Zaurus de Sharp ou bien le téléphone Nokia 9210i. Une liste de produits est disponible sur *http://www.opera.com/embedded/products.* Des versions de démonstration du navigateur Opera/Embedded sont également disponibles en ligne mais il est nécessaire d'obtenir un code d'accès auprès de l'éditeur.

En résumé

La majorité des applications embarquées sous Linux utilisent le mode texte. La console texte Linux peut tout à fait être configurée tant au niveau des polices de caractères que des claviers supportés.

X Window System est le système graphique de prédilection de Linux. La version XFree86-4 peut être optimisée pour un environnement embarqué en utilisant les fonctions VESA des chipsets graphiques. L'utilisation du frame-buffer permet d'accéder directement à des modes graphiques haute résolution à travers le noyau Linux sans forcément utiliser X. D'autres bibliothèques graphiques dédiées comme Qt/Embedded, GTK-Embedded ou MicroWindows/Nano-X sont parfois mieux adaptées lorsque l'empreinte mémoire nécessaire est faible. D'autres composants comme LCDproc sont disponibles dans le cas où l'interface se réduit à la gestion d'un afficheur de petite taille comme un LCD.

La technologie web basée sur le protocole HTTP et le langage HTML peut être très intéressante pour un système embarqué configurable à distance. Il faut alors l'équiper d'un programme serveur HTTP adapté. L'utilisation d'un navigateur web embarqué peut se révéler complexe et coûteuse si le système doit être compatible avec les extensions multimédias des navigateurs classiques.

Quatrième partie

Études de cas

Cette dernière partie comprend deux études de cas reposant sur Linux et issues du monde industriel. La première étude concerne un lecteur/enregistreur MP3 de salon entièrement fait de composants Open Source. La deuxième étude présente un terminal de consultation Internet fondé sur une version adaptée de Netscape Communicator 4.79. Outre leur fonction initiale, ces deux projets permettent de décrire quelques techniques intéressantes communément utiles dans le monde de l'embarqué.

Les composants et méthodes traités dans ces études sont naturellement réutilisables dans d'autres projets industriels.

13

Open Music Machine

Description du projet

Le projet *Open Music Machine* (OMM) a pour objet de développer une platine de lecture/enregistrement évolutive, basée sur un système ouvert. Il s'agit de réaliser un système simple et convivial dont l'interface utilisateur sera similaire à celle d'une platine CD classique, toute la partie « informatique » devant être masquée à l'utilisateur. Le pilotage du système est entièrement réalisé par une télécommande infrarouge et l'affichage est effectué sur un écran LCD de faible dimension (4 lignes de 20 caractères). Ce projet est typique d'une application embarquée et de l'utilisation de composants Open Source. Différentes parties du projet, comme l'API gestion des actions utilisateur ou la gestion de l'affichage LCD, sont réutilisables dans de nombreux projets similaires, tout le code étant diffusé sous GPL.

L'idée initiale est d'utiliser des composants standards et peu coûteux (architecture x86 « grand public », système d'exploitation Linux, composants Open Source). Les principales fonctions envisagées pour OMM sont les suivantes :

- lecture des CD audio avec affichage des informations par l'accès à une base de données de type CDDB (*http://www.freedb.org*, en cas de connexion réseau disponible)
- lecture des fichiers MP3
- création de fichier MP3 à partir de fichiers audio extraits des CD
- lecture de CD MP3
- outil de navigation (sélecteur de fichiers)
- télécommande à infrarouge et gestion d'afficheur LCD 20×4

- connexion réseau de type Ethernet (câble ou ADSL)
- client NAPSTER embarqué
- système d'exploitation multi-tâche permettant de basculer d'une fonction à l'autre lorsque c'est possible (par exemple : utiliser le sélecteur de fichiers tout en écoutant un morceau).

D'autres fonctions, comme la gravure de CD-Rom MP3 ou de CD audio, sont facilement envisageables. Un mode de connexion réseau plus bas de gamme de type MODEM ou ISDN pourrait également être envisagé car nous avons vu au chapitre 6 que la mise en place d'une connexion PPP sous Linux était simple et peu onéreuse en espace disque. Une des caractéristiques du produit est la forte modularité et nous verrons plus tard que les fonctions sont réalisées par de petits modules indépendants eux-mêmes gérés par un module appelé *manager*. En ce qui concerne l'architecture matérielle, il est également logique d'envisager plusieurs déclinaisons du produit :

- Une version minimale permettant de lire des CD audio ou MP3 mais ne disposant pas de disque dur. Ce dernier est remplacé par un disque flash dont la taille ne permet que de stocker de très faibles volumes (quelques fichiers MP3). Cette configuration se rapproche de celle des baladeurs MP3 qui utilisent de la mémoire flash (souvent au format *CompactFlash*) pour stocker les fichiers MP3. Il est d'ailleurs intéressant de noter que ces baladeurs utilisent parfois de vrais systèmes d'exploitation embarqués industriels (cas du baladeur MP3 IOMEGA utilisant le système libre).
- Le disque dur permet bien sûr d'ajouter la fonction de stockage et donc de gestion d'un catalogue de fichiers. Pour des raisons de compatibilité entre les versions matérielles, le système d'exploitation (Linux) est systématiquement stocké sur la mémoire flash, même en cas de présence de ce disque.
- La connexion réseau est optionnelle car elle ne concerne que la fonction d'identification des CD audio et celle du client NAPSTER (téléchargement de fichiers MP3).

Pour des raisons de coût, toutes les fonctions sont réalisées par des composants logiciels. Ce choix présente quelques inconvénients au niveau des performances (décompression ou compression MP3) mais permet une grande souplesse car un nouveau format ou une nouvelle fonction peuvent être ajoutés très facilement.

Lors du démarrage du projet (en mai 2000), il existait déjà plusieurs projets similaires ou même des produits industriels. À de rares exceptions près, ces projets existants ne semblaient pas satisfaisants car souvent trop proches d'un système informatique ou bien trop proches d'une platine CD classique. La présence d'une connexion réseau est extrêmement rare dans les produits de ce type alors qu'elle est à la base de l'utilisation de fonctions communicantes. Dans le cas de produits basés sur des composants matériels spécifiques, l'absence de système d'exploitation évolué est la raison principale de cette limitation, ainsi que celle du manque d'évolutivité du produit. Ce dernier point correspond également à une réalité économique (inciter le consommateur à remplacer son matériel) qui n'entrait pas en ligne de compte pour le projet OMM.

Organisation du projet

Le projet a été développé par deux personnes géographiquement « éloignées ». L'une fut chargée des aspects applicatifs, l'autre ayant la responsabilité des aspects matériels et gestion des composants de type télécommande infrarouge, afficheur LCD ou système d'exploitation. Une seule version complète du matériel était disponible (et pas immédiatement) et il fut donc nécessaire de mettre en place une solution d'émulation permettant de développer la partie applicative sans disposer d'un matériel définitif. En optant pour un tel mode de fonctionnement, on peut aussi accélérer le temps de développement et améliorer la modularité du code. Pour passer d'un environnement de développement à l'environnement réel, il suffit de compiler les applications avec les pilotes matériels adéquats.

Les différents environnements de développement furent les suivants :

- Environnement de développement sous X11, l'afficheur LCD étant émulé dans une fenêtre X. La partie télécommande infrarouge est simulée par une application graphique rappelant la géométrie de celle-ci.
- Environnement de développement sous Curses, l'afficheur étant simulé par une fenêtre en mode texte et la partie télécommande par de simples raccourcis clavier.
- Environnement réel utilisant les véritables pilotes infrarouges et LCD.

Chaque environnement est associé à une bibliothèque `dpy_HW.a` (`dpy_x11.a`, `dpy_curses.a`, `dpy_lcd.a`) qui contient les fonctions de gestion des événements utilisateur et de l'affichage LCD. Des informations générales concernant les interfaces graphiques embarquées sont disponibles au chapitre 12.

Dans le cas du développement dans l'environnement X11, l'affichage LCD a l'allure suivante.

Figure 13-1
Afficheur LCD sous X11.

Architecture globale

Le système est basé sur une version réduite de Linux similaire à celle décrite au chapitre 5. Dans ce type de système, le temps de démarrage est primordial car l'utilisateur n'est pas habitué à un système informatique mais à un appareil électronique qui n'utilise pas de système d'exploitation évolué et donc qui est opérationnel dès la mise sous tension. Un système réduit du type de celui décrit au chapitre 5 a généralement un temps de

démarrage inférieur à dix secondes, auquel il faut cependant ajouter le temps d'initialisation du BIOS de la carte, sur lequel nous avons très peu de possibilités de configuration. Le cumul de ces deux délais peut parfois devenir critique et c'est la raison pour laquelle la version finale d'OMM utilise LinuxBIOS décrite au chapitre 8. Grâce à LinuxBIOS, le temps de mise à disposition du système est réduit à quelques secondes mais cela oblige à utiliser une carte mère compatible.

En raison des choix techniques et des contraintes mentionnées au début de ce chapitre, l'architecture se devait d'être extrêmement modulaire. Il n'était pas envisageable de réaliser un logiciel unique chargé de toute la partie applicative. De même, toutes les couches logicielles devaient être clairement séparées afin de permettre l'utilisation des différents environnements de développement conduisant systématiquement à la génération de trois versions d'exécutables (X11, Curses et exécutables réels). L'architecture globale du système est décrite de façon schématique dans le diagramme qui suit :

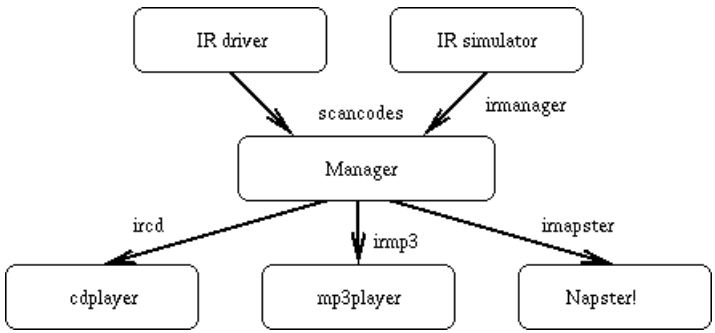


Figure 13-2

Diagramme de l'architecture globale.

Les composants sont divisés en trois catégories :

- Les composants applicatifs graphiques, qui effectuent des affichages sur le LCD. Ils correspondent à la partie visible des différentes fonctions du système (lecteur CD, lecteur MP3, encodeur MP3, sélecteur de fichiers).
- Les composants externes, qui n'utilisent pas d'interface graphique et qui sont en général issus de composants Open Source modifiés pour OMM. Un exemple en est le célèbre décodeur MP3 mpg123. Ces programmes sont pilotés par les composants graphiques précédemment cités.
- Le programme manager qui est chargé de recevoir les actions utilisateur provenant de la télécommande infrarouge, ou bien de son simulateur, et de répartir les ordres aux différents composants graphiques.

La communication entre ces différents modules utilise un composant Unix appelé « tube nommé » (*named pipe* ou *FIFO*). Ce composant permet de mettre en place très simplement un schéma de type fournisseur/consommateur, ce qui explique l'analogie

avec le « tube » Unix (ou *pipe*) largement utilisé dans toutes les versions d'Unix dont, bien sûr, Linux.

Rappel

Le tube (symbolisé par la barre verticale) est un composant qui permet de chaîner des commandes Unix afin que la sortie d'une commande devienne l'entrée de la commande suivante. On pourra par exemple compter le nombre de fichiers d'un répertoire en faisant `ls -l | wc -l`.

La différence entre un tube et un tube nommé réside dans le côté fugitif du tube classique qui n'existe que le temps de l'exécution de la commande. Dans le cas du tube nommé, ce dernier est associé à un fichier spécial comme indiqué ci-après. Nous remarquons que le premier caractère des attributs du fichier est la lettre *p* comme *pipe*.

```
$ ls -la /tmp/ircd*
prw-r--r--    1 pierre   users       0 jan 29  2001 /tmp/ircd
prw-r--r--    1 pierre   users       0 jan 29  2001
➥/tmp/ircd_reply
```

Comme le montre la liste ci-après, le principe a consisté à utiliser deux tubes nommés par fonction, l'un étant destiné à la réception des commandes et l'autre à la réponse, si nécessaire. Pour envoyer une commande à un module, il suffit donc d'écrire le caractère associé sur le tube comme ci-après :

```
echo -n p > /tmp/ircd
```

qui démarre le mode lecture (*play*) sur le module de lecture de CD audio. Au niveau de l'interface de programmation, un tube nommé sera facilement créé par les quelques lignes de code qui suivent :

```
int open_fifo (char *fifo_name)
{
  int fd;

  /* Create if not existing */
  if (mknod (fifo_name, S_IFIFO | 0666, 0) != 0  && errno != EEXIST) {
    log_err ("%s: %s", fifo_name, strerror(errno));
    exit (1);
  }

  if ((fd = open (fifo_name, O_RDWR)) < 0) {
    log_err ("%s: %s", fifo_name, strerror(errno));
    exit (1);
  }

  return fd;
}
```

Le descripteur ainsi calculé pourra ensuite être utilisé comme un descripteur de fichier classique. Comme indiqué sur le diagramme, le programme **`manager`** utilise des tubes

nommés (irmp3, ircd, etc.) afin de répercuter les ordres de l'utilisateur aux différents modules concernés. Les programmes graphiques ainsi que le manager sont gérés par une petite bibliothèque événementielle permettant de manipuler un automate à états finis. La réception d'une commande (un caractère) dans un état donné est associée à l'exécution d'une fonction et au passage dans un nouvel état comme le montre l'exemple exposé ci-après :

```
static omm_event_handler cd_event_handlers[] = {
  'e', STATE_CD, cd_player_eject_close,         /* Open/close tray */
  'e', OMM_STATE_INIT, cd_player_eject_close,   /* Open/close tray */
  'p', STATE_CD, cd_player_play,                /* Play */
  'p', OMM_STATE_INIT, cd_player_play,          /* Play */
  's', STATE_CD, cd_player_stop,                /* Stop */
/* ... */
OMM_DEFAULT_EVENT, OMM_ANY_STATE, cd_player_default_handler,
  0, 0, NULL
};
```

L'état initial de l'automate est **OMM_STATE_INIT**. Si le programme détecte un CD dans le lecteur, il passera dans l'état **STATE_CD**. En cas de réception du caractère *e* sur la FIFO de commande, le plateau du lecteur sera alternativement ouvert ou fermé par appel à la fonction **cd_player_eject_close()**, et l'état passera donc alternativement de **OMM_STATE_INIT** à **STATE_CD**. De même, le démarrage de la lecture du CD par réception de *p* (play) ne pourra se faire que si un CD est présent (**STATE_CD**), et ce par appel à la fonction **cd_player_play()**.

La partie principale du programme initialise l'API en lui passant la table des événements et le nom du tube nommé à utiliser. Le troisième paramètre est une valeur booléenne indiquant la possibilité de pouvoir piloter l'application depuis l'entrée standard (touches du clavier) et non seulement depuis le tube nommé.

```
  /* Init API */
  omm_api_init (cd_event_handlers, CDPLAYER_FIFO, use_stdin);
  lcd_clear_screen ();

  /* ... */

  /* Main loop */
  omm_api_main_loop ();

  /* End */
  log_debug (DEBUG_CDPLAYER, "exiting.");
  close_debug ();
```

Pour le test en environnement de développement, une solution simple et élégante consiste à développer un petit simulateur de télécommande en utilisant le langage *Perl-Tk*. Ce langage permet de construire rapidement des applications graphiques en recourant à la syntaxe du langage Perl (*http://www.perl.org*). Chaque touche affichée simule une touche de la télécommande et écrit donc un caractère particulier sur le tube nommé d'entrée du programme manager. En fonction de l'état de l'automate du manager, la commande en

question est ensuite exécutée. La figure ci-après représente l'interface de simulation de la télécommande.

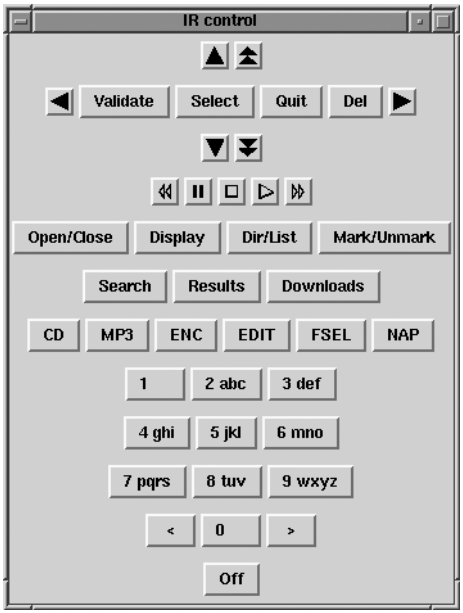


Figure 13-3

Simulation de la télécommande.

Outre l'API de gestion des événements, l'API d'affichage sur le LCD doit également être portable sur les trois environnements cible. La base de cet affichage est la fonction `lcd_write()` qui est indépendante de la cible. La fonction `lcd_write()` utilise elle-même la fonction `lcd_hw_write()` qui elle est définie en fonction de l'environnement matériel de la cible. Dans le cas de l'environnement X11, cette dernière fonction se termine par un appel à la fonction `XdrawImageString()` de la bibliothèque `Xlib` alors que, pour la cible Curses, on utilise la fonction classique `mvprintw()`.

Utilisation des composants externes

OMM utilise exclusivement des composants Open Source et l'objectif était de développer le moins de code particulier possible et donc de s'appuyer sur des bibliothèques ou programmes existants. Cette approche a un but pédagogique évident, démontrant que l'Open Source peut réduire sensiblement le délai de développement d'un produit. La liste des composants externes est brièvement listée ci-après :

- *libcdaudio* (*http://cdcd.undergrid.net/libcdaudio*) pour l'accès au lecteur de CD audio ainsi que la gestion des requêtes à la base de données CDDB

- *mpg123* (*http://www.mpg123.de*) pour la lecture des fichier MP3
- *cdparanoia* (*http://www.xiph.org/paranoia*) pour l'extraction des pistes d'un CD audio
- *gogo* (*http://homepage1.nifty.com/herumi/gogo_e.html*), *bladeenc* (*http://bladeenc.mp3.no*) ou *lame* (*http://www.sulaco.org/mp3*) pour l'encodage MP3
- *nap* pour le client NAPSTER (*http://nap.sourceforge.net*)
- *zgsmplay* pour la base du sélecteur de fichiers (*http://rus.members.beeb.net/zgsmplay.html*).

À l'exception de libcdaudio qui est une bibliothèque, les autres composants sont des programmes indépendants qui sont utilisés tels quels ou très peu modifiés. La principale modification apportée est l'ajout ou l'amélioration d'une fonction de « pilotage à distance » du programme depuis son entrée standard. Des fonctions de ce type sont déjà disponibles sur des programmes comme mpg123 car leur pilotage à distance est assez couramment utilisé.

Cette méthode permet de lancer la commande depuis l'applicatif graphique par un `fork()` suivi d'un appel à `execlp()` comme l'indique cet extrait du code du module de lecture MP3 :

```
if (pipe (pipe_in) < 0) {
   log_err ("pipe_in: %s", strerror(errno));
   exit (1);
 }

 if (pipe (pipe_out) < 0) {
   log_err ("pipe_out: %s", strerror(errno));
   exit (1);
 }

 if (!(mpg123_pid = fork ())) {
   dup2 (pipe_out[0], 0);
   close (pipe_out[0]);
   dup2 (pipe_in[1], 1);
   close (pipe_in[1]);
   close (2);

   if (execlp (MPG123, MPG123, "-R", "-", NULL) < 0) {
     log_err ("execlp: %s: %s", MPG123, strerror(errno));
     exit (1);
   }
 }
```

Les options « `-R -` » indiquent à mpg123 de travailler en mode pilotage à distance, ce qui signifie qu'il exécutera les différentes actions en fonction des commandes reçues sur son entrée standard et qu'il affichera le résultat sur sa sortie standard. Ce principe est très pratique ici car l'entrée et la sortie standard du programme sont redirigées sur des tubes initialisés lors de la création du nouveau processus associé à mpg123. Dans l'exemple qui suit, le dialogue avec mpg123 est tracé en mode pilotage à distance. Dans ce cas précis,

le programme est lancé par un utilisateur dans un interpréteur de commande. Par convention, les messages envoyés par le programme piloté commencent par le caractère @.

```
$ mpg123 -R -
@R MPG123
L Eagles - Hotel California (Live Version).mp3
@I Eagles - Hotel California (Live Version)
@S 1.0 3 44100 Joint-Stereo 0 365 2 0 1 0 112 0
@F 0 15804 0.00 412.84
@F 1 15803 0.03 412.81
@F 2 15802 0.05 412.79
@F 4 15801 0.10 412.76
@F 4 15800 0.10 412.73
...
@P 1
...
$
```

Au lancement du programme, celui-ci affiche son identification. Cela peut être mis à profit pour vérifier la version du programme. La commande *L* (*load*) tapée par l'utilisateur indique au programme le nom du fichier MP3 à jouer ; mpg123 retourne alors la ligne des *ID3 tags* contenant les caractéristiques du morceau suivies des paramètres de l'encodage. Ces paramètres pourront bien sûr être formatés et affichés sur l'écran du LCD. La suite des lignes *@F* constitue la progression de la lecture et sera également affichée. Si l'utilisateur tape le caractère *p* (*pause/play*), le programme répond *@P* qui indique l'état de pause. En envoyant de nouveau un *p*, on poursuit la lecture jusqu'à la fin du fichier ou bien jusqu'à la réception du caractère *q* (*quit*).

Détail des API

Gestion des événements

La bibliothèque de gestion des événements est très simple, la taille du fichier principal **`omm_api.c`** étant inférieure à 80 lignes. Comme nous avons pu l'indiquer précédemment, la gestion des événements est basée sur une structure définissant des pointeurs de fonctions associés à des actions opérateur et à l'état de l'automate. Cette API est intimement liée à l'API de gestion de l'affichage décrite au paragraphe suivant. Les états sont définis par la structure suivante :

```
typedef struct _omm_event_handler {
  int code;    /* event code */
  int current_state;  /* state before event */
  void (*handler)();  /* event handler */
} omm_event_handler;
```

Les fonctions associées sont les suivantes :

```
void omm_api_init (omm_event_handler *event_handler_tab, char *fifo_name,
➥ boolean use_stdin)
```

Cette fonction initialise l'interface à partir de la table des événements **tab** et du nom du tube nommé **fifo_name** recevant les actions de l'opérateur. La valeur booléenne **use_stdin** indique si l'application peut recevoir des actions simulées depuis l'entrée standard *stdin* (utilisée pour le test).

```
void omm_api_close (void)
```

ferme l'interface. De façon typique, ferme le tube nommé de communication.

```
void omm_api_get_event (omm_api_event *ev)
```

lit l'événement en attente.

```
void omm_api_handle_event (omm_api_event *ev)
```

traite l'événement lu précédemment. En l'occurrence, exécute la fonction associée selon l'état de l'automate.

```
void omm_api_main_loop (void)
```

boucle infinie principale de gestion des événements. Cette fonction utilise les deux fonctions précédentes comme suit :

```
  while (!omm_api_end) {
    omm_api_get_event (&ev);
    omm_api_handle_event (&ev);
  }
omm_api_event omm_api_get_current_event ()
```

retourne la valeur de l'événement courant.

```
void omm_api_start_input (char *s, void (*fn_ok), void (*fn_abort));
```

initialise une saisie de l'opérateur. La valeur sera stockée dans la chaîne de caractères **s**. La fonction associée au pointer **fn_ok** est exécutée en cas de validation de la saisie. La fonction associée au pointer **fn_abort** est exécutée en cas d'annulation de la saisie.

```
char *omm_api_get_input_str ();
```

retourne la valeur de la chaîne saisie.

Gestion de l'écran LCD

La bibliothèque de gestion de l'écran présente une couche indépendante du matériel associée à une couche de bas niveau liée à ce matériel (**lcd_hw**). Les sources de chaque bibliothèque d'affichage sont définies dans un fichier **dpy_HW.c**, *HW* correspondant au type d'interface matérielle, soit :

- **lcd** pour l'afficheur LCD réel
- **x11** pour l'émulation X11
- **curses** pour l'émulation Curses

Le fait d'ajouter un nouveau périphérique matériel (nouveau type d'écran ou nouvelle couche d'émulation) se réduit à définir les fonctions suivantes :

```
void lcd_hw_init (boolean use_stdin)
```

Cette fonction effectue l'initialisation matérielle du périphérique d'affichage. Dans le cas de l'écran LCD réel, cette partie contient en général des accès au port de contrôle du périphérique (souvent, port série ou port parallèle), comme l'indique l'extrait de code ci-après :

```
  if(ioperm(PORTADDRESS,3,1)) {perror("ioperm"); exit(1);}

  pthread_mutex_init (&write_mutex, NULL);
  outb(inb(CONTROL) & 0xDF, CONTROL);
  outb(inb(CONTROL) | 0x08, CONTROL);
  for (count = 0; count <= NB_COMMAND ; count++) {
    outb(init[count], DATA);
    outb(inb(CONTROL) | 0x01, CONTROL);
    usleep(2);
    outb(inb(CONTROL) & 0xFE, CONTROL);
    usleep(2);
  }
  outb(inb(CONTROL) & 0xF7, CONTROL);
/* ... */
```

Notez l'appel à la fonction **`ioperm()`** qui permet à un programme en mode utilisateur d'accéder à des ressources normalement réservées aux pilotes de périphériques. Cette solution est à utiliser avec précaution et il est en général préférable à terme de développer un pilote.

Dans le cas de l'émulation X11 ou Curses, le code correspond à l'initialisation de l'interface et de la fenêtre, comme montré ci-après pour la version Curses :

```
  initscr ();
  clear ();
  crmode ();
  noecho ();
  curs_set (0);

  xref = (COLS - 22)/2;
  yref = y = 3;

  mvprintw (y++, xref, ".--------------------.");
  mvprintw (y++, xref, "|                    |");
  mvprintw (y++, xref, "|                    |");
  mvprintw (y++, xref, "|                    |");
  mvprintw (y++, xref, "|                    |");
  mvprintw (y++, xref, "`--------------------'");
  refresh ();

  curses_use_stdin = use_stdin;

void lcd_hw_close (void);
```

Cette fonction ferme l'interface. L'exemple de Curses présenté ci-après indique un retour au mode d'écran normal :

```
    tcsetattr(fileno(stdin), TCSANOW, &old_stdin);
    endwin ();

void lcd_hw_write (int x, int y, char *s);
```

Cette fonction écrit une chaîne de caractères `s` aux coordonnées `x` et `y` de l'écran. Elle est à la base de toutes les fonctions publiques d'affichage.

```
int lcd_hw_input (void);
```

lit un caractère venant de l'opérateur (tube nommé). Dans le produit final, cela correspond aux actions de la télécommande. Dans ce cas, la télécommande est gérée par un programme indépendant qui lit les trames infrarouges (venant en général d'un port série), et formate la sortie en caractères identiques à ceux envoyés par le simulateur `xir.pl`.

```
unsigned long lcd_hw_get_window(void);
```

retourne un pointeur vers la fenêtre d'affichage.

Lorsque ces fonctions sont définies, l'API de gestion de l'affichage fournit les fonctions suivantes, définies dans le fichier `lcd.c` :

```
void lcd_init (boolean use_stdin)
```

initialise l'interface d'affichage (appel de `lcd_hw_init`)

```
void lcd_close (void)
```

ferme l'interface d'affichage (appel de `lcd_hw_close`)

```
void lcd_dump_screen (void)
```

affiche le contenu de l'écran dans le fichier de trace. Cette fonction est utilisée dans les phases de test et de mise au point.

```
void lcd_hide_screen (void)
```

masque l'affichage de l'écran courant. Cette fonction est utile pour passer une application en « arrière-plan » (*background*) comme cela est décrit plus bas dans le chapitre. L'affichage d'une fonction en arrière-plan ne s'effectue plus directement sur l'écran mais dans un tableau de caractères.

```
void lcd_restore_screen (void)
```

retour de l'application en avant-plan (*foreground*). Le contenu du tableau est affiché.

```
int lcd_input ()
```

lit un caractère provenant de l'opérateur (appel de `lcd_hw_input`).

```
void lcd_write (int x, int y, char *s, lcd_attr attr)
```

affiche la chaîne de caractères `s` aux coordonnées `x` et `y` de l'écran en utilisant l'attribut `attr`.

Les seuls attributs gérés sont actuellement l'attribut d'affichage normal (**ATTR_NORMAL**) et clignotant (**ATTR_BLINK**).

```
void lcd_write_center(int y, char *s, lcd_attr attr)
```

affiche la chaîne **s** centrée sur la ligne **y** de l'écran avec l'attribut **attr**.

```
void lcd_clear(int x1, int y1, int x2, int y2)
```

efface une zone de l'écran définie par deux points (x1, y1) et (x2, y2).

```
void lcd_clear_screen(void)
```

efface entièrement l'écran.

```
void lcd_clear_line(int y)
```

efface la ligne y.

```
void lcd_init_hscroll (void)
```

initialise l'affichage d'une chaîne de caractères de longueur supérieure à la largeur de l'écran.

```
void lcd_write_hscroll (int x, int y, char *s)
```

affiche une chaîne de caractères **s** de longueur supérieure à la largeur de l'écran aux coordonnées **x** et **y**. L'affichage est réalisé par un *scrolling* horizontal.

```
void lcd_move (int x, int y)
```

déplace la position courante d'affichage en **x** et **y**.

```
void lcd_addch (int c, lcd_attr attr)
```

affiche un caractère à la position d'affichage courante.

```
void lcd_addstr (char *s, lcd_attr attr)
```

affiche la chaîne **s** à la position d'affichage courante.

```
unsigned long lcd_get_window ()
```

retourne la valeur du pointeur de fenêtre d'affichage (appel de **lcd_get_hw_window**).

Arborescence des sources et compilation des modules

Les sources sont organisés de la manière suivante :

```
-rw-r--r--    1 pierre   users         785 Dec 12  2000 Makefile
-rw-r--r--    1 pierre   users         219 Dec 11  2000 README
drwxr-xr-x    6 pierre   users        4096 Jan 28  2001 auxprogs
drwxr-xr-x   11 pierre   users        4096 Mar  6 13:44 ommprogs
drwxr-xr-x    4 pierre   users        4096 Dec 29  2000 xir
```

Le répertoire **auxprogs** contient les programmes auxiliaires non graphiques récupérés à l'extérieur. Le répertoire **ommprogs** contient tous les modules graphiques ainsi que les API

susmentionnées (répertoire **common**). Le fichier **Makefile** de la racine utilise une fonction récursive afin d'explorer tous les sous-répertoires.

```
SUBDIRS= ommprogs auxprogs

all::
  for i in  $(SUBDIRS) ;\
  do \
  (cd $$i ; echo "making all" "in $$i..."; \
  $(MAKE) $(MFLAGS) all); \
  done
```

Les API utilisant un environnement multi-cible, il est intéressant de définir le fichier **Makefile** de manière à générer automatiquement les différentes versions suffixées par le nom de la cible x11, lcd ou curses. L'exemple présenté ci-après décrit le fichier **Makefile** du module cdplayer.

```
LIBDPYCURSES= libdpy_curses.a
EXTRALIBSCURSES= -lcurses
LIBDPYX11= libdpy_x11.a
EXTRALIBSX11= -L/usr/X11R6/lib -lX11
LIBDPYX11= libdpy_lcd.a
EXTRALIBSLCD=
MISCLIBS= -lcommon -lcdaudio -lpthread

SRCS= cdplayer.c

OBJS= cdplayer.o

all: cdplayer_curses cdplayer_lcd cdplayer_x11

cdplayer_curses: $(OBJS) ../common/$(LIBDPYCURSES) ../common/libcommon.a
  $(CC) $(CFLAGS) -o $@ $(OBJS) -L../common -ldpy_curses $(EXTRALIBSCURSES)
  $(MISCLIBS)

cdplayer_x11: $(OBJS) ../common/$(LIBDPYX11) ../common/libcommon.a
  $(CC) $(CFLAGS) -o $@ $(OBJS) -L../common -ldpy_x11 $(EXTRALIBSX11) $(MISCLIBS)

cdplayer_lcd: $(OBJS) ../common/$(LIBDPYLCD) ../common/libcommon.a
  $(CC) $(CFLAGS) -o $@ $(OBJS) -L../common -ldpy_lcd $(EXTRALIBSLCD) $(MISCLIBS)
```

Description des différents modules

Module de gestion des fonctions (manager)

Le manager est le noyau central de la gestion des fonctions d'OMM. Ce module reçoit systématiquement les ordres de la télécommande *via* un tube nommé et redistribue les ordres aux différents modules en fonction de l'état de l'automate. Le premier rôle imparti à ce module est de sélectionner la fonction active en appuyant sur l'une des six touches de fonction CD (lecteur de CD audio), MP3 (lecteur MP3), ENC (encodage de pistes

audio en compilations MP3), EDIT (édition des compilations), FSEL (sélecteur de fichier), NAP (module NAPSTER). Lorsqu'un des modules est sélectionné, un tube nommé par défaut est utilisé pour diriger les actions de l'utilisateur vers l'application courante. La touche *Quit* de la télécommande permet de quitter l'application courante.

Certains modules d'OMM peuvent coexister à un instant donné, une application apparaissant en premier plan (*foreground*) et les autres en arrière-plan (*background*). L'application qui se trouve en premier plan est celle qui affiche les informations directement sur le LCD. Située en arrière-plan une application affichera les informations dans un écran « virtuel » qui sera copié dans le LCD si l'application revient au premier plan. Une illustration simple en est la possibilité d'utiliser le sélecteur de fichier tout en écoutant un CD audio ou bien un fichier MP3.

Module de lecture des CD audio

Le module de lecture CD cdplayer est basé sur la bibliothèque libcdaudio. Cette bibliothèque diffusée sous GPL est très simple à utiliser et elle est à la base d'utilitaires de lecture de CD sous Linux comme demcd.

Une fois le lecteur CD ouvert, celui-ci est associé à un descripteur de fichier auquel on peut appliquer toutes les fonctions disponibles.

```
if ((fd_cd = cd_init_device (CD_DEV)) < 0) {
  log_err ("Can't init device %s, exiting.\n", CD_DEV);
  exit (1);
}
```

On peut ensuite tester la présence du CD :

```
  int fd_cd;
  struct disc_info disc_info;

  if (cd_stat (fd_cd, &disc_info) < 0)
    log_err ("Can't stat CD");

  if (disc_info.disc_present) {
     /* ... */
  }
```

puis effectuer des opérations classiques comme la lecture, l'arrêt, la pause ou l'éjection du CD.

```
if (cd_play (fd_cd, current_track) < 0)
    log_err ("Can't play CD");

/* ... */

if (cd_stop (fd_cd) < 0)
  log_err ("Can't stop CD");
```

```
/* ... */

if (cd_pause (fd_cd) < 0)
  log_err ("Can't pause CD");

/* ... */

if (cd_eject (fd_cd) < 0)
    log_err ("Can't eject CD");
```

Ce module doit avoir les fonctions habituelles d'un lecteur de CD classique (voir les touches de la télécommande) mais doit également pouvoir gérer les fonctions spécifiques à OMM comme l'interrogation de la base CDDB ou bien le marquage de pistes audio pour un encodage MP3 ultérieur. Dans le cas d'un fonctionnement compatible avec un lecteur classique, l'affichage a l'apparence suivante :

Figure 13-4
Lecteur de CD audio.

L'afficheur LCD présente des informations classiques comme le numéro de la piste, sa durée et sa position courante. On peut également afficher la durée totale du CD. Si OMM est capable d'interroger la base CDDB, il pourra récupérer des informations plus complètes comme le titre de l'album, le nom de l'artiste et les titres des différentes pistes. Sur la base CDDB, les informations sont indexées par une valeur numérique (*Disc ID*) codant de manière unique le CD en cours d'utilisation. Pour limiter les accès à la base, ces informations sont ensuite stockées dans un espace « cache » sur le disque dur local. L'accès à la base CDDB est possible depuis la bibliothèque libcdaudio. À partir du calcul de l'identifieur du CD, on peut lire l'information à partir du réseau puis l'écrire sur le disque local.

```
int disc_id;
struct disc_data disc_data;

disc_id = cddb_direct_discid (disc_info)
if (cddb_read_disc_data (fd_cd, &disc_data) < 0)
   log_debug (DEBUG_CDPLAYER, "Can't access CDDB for %x!", disc_id);
else {
   if (cddb_write_disc_data (fd_cd, disc_data) < 0)
      log_err ("Can't write CCDB !");
      /* ... */
}
```

Une touche de la télécommande (*Display*) permet de commuter sur l'affichage de ce type d'informations si elles sont disponibles. L'afficheur présente alors les informations suivantes :

Figure 13-5
Affichage d'informations CDDB.

L'afficheur LCD étant de petite taille en largeur (seulement 20 caractères), la bibliothèque de gestion du LCD effectue un *scrolling* en temps réel.

Comme il est également possible au moyen d'OMM d'encoder des pistes audio au format MP3, le module de lecture CD permet donc de « marquer » les pistes qui seront ensuite compressées par le module d'encodage *encoder*. Le marquage est effectué par la touche *Mark/Unmark* de la télécommande. Dans le cas de la sélection d'une piste, l'afficheur a recours à un indicateur comme cela est décrit ci-après :

Figure 13-6
Sélection d'une piste.

Il faut noter que l'indicateur de sélection d'une piste est stocké dans le système, cette piste étant repérée de manière unique par l'identifieur de CD et le numéro de piste. On peut cependant annuler la sélection de la piste en la sélectionnant de nouveau, ce qui annule l'affichage de l'indicateur.

Module de navigation/sélection de fichiers

Le sélecteur de fichier permet de manipuler les fichiers MP3 générés à partir des pistes audio ou bien téléchargés *via* le réseau. Grâce à cet utilitaire, on peut sélectionner un ensemble de fichiers qui seront ensuite lus par le module de lecture MP3 mp3player décrit plus bas. Il faut noter que ce module peut être exécuté en parallèle avec d'autres modules comme la lecture CD ou MP3. On peut en effet manipuler des fichiers tout en écoutant un CD ou des fichiers MP3.

Le module de sélection de fichier fileselector est très important pour un produit de type OMM car le fait de pouvoir générer un grand nombre de fichiers MP3 nécessite donc de disposer d'un outil pratique pour manipuler ces fichiers, sans oublier que l'interface d'affichage et de pilotage est rudimentaire par rapport à un système informatique (LCD 20×4 et télécommande). Le problème qui se pose est typique d'un système embarqué, en l'occurrence fournir une interface fonctionnelle conviviale dans un environnement le plus réduit possible. Développer un outil de ce type n'est pas forcément simple et le programme est donc fortement inspiré de la partie sélection de fichiers de l'utilitaire zgsmplay destiné à la lecture de fichier audio au format GSM *via* une interface texte basée sur Curses.

Pour simplifier l'interface, OMM utilise un seul niveau de répertoire qui correspond aux noms des compilations créées par le module d'encodage MP3 décrit plus avant. En mode sélection de fichiers, l'affichage a l'apparence suivante :

Figure 13-7
Sélecteur de fichiers.

Afin de permettre l'utilisation d'un afficheur LCD quelconque, le curseur est entièrement géré par la bibliothèque OMM. Il est représenté par le caractère > . La lettre *D* en début de ligne indique un répertoire dans lequel on pourra entrer en appuyant sur la flèche droite de la télécommande. On retourne au niveau supérieur en appuyant sur la flèche gauche. Lorsqu'on explore une liste de fichiers, on peut sélectionner un fichier ou un répertoire entier en appuyant sur la touche de validation (*Validate*) de la télécommande.

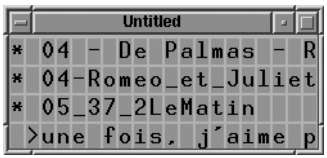


Figure 13-8
Fichiers sélectionnés.

Outre la sélection de fichiers, OMM est également capable de manipuler les *playlists* contenant une suite de fichiers MP3 construite à partir de diverses compilations. On peut passer du mode fichier au mode *playslist* en appuyant sur la touche *Dir/List* de la télécommande.

Modules de lecture MP3

Le module de lecture MP3 mp3player est une interface au programme mpg123, utilisée en mode pilotage à distance, comme décrit au début de ce chapitre. L'avantage de ce mode de fonctionnement est double. Il permet en premier lieu de s'affranchir de l'écriture d'un programme complexe et de se concentrer sur la valeur ajoutée du produit que l'on développe. Conséquence corollaire, on peut profiter des évolutions du produit sans modifier son propre logiciel. Le deuxième point important est la possibilité de modifier le module en effectuant des retouches mineures sur le logiciel. Dans le cas présent, il ne faut pas oublier que le format MP3 est soumis à des brevets (*patent*) et qu'il n'est en théorie pas légal d'utiliser mpg123 dans un produit industriel dans de nombreux pays. L'utilisation d'un produit commercial officiel comme *xaudio* permet de résoudre le problème en effectuant dans ce cas-là aussi des modifications raisonnables.

Le principe consiste à sélectionner une liste de fichiers MP3 à jouer grâce au sélecteur de fichiers. Si l'on sélectionne le module MP3 sur la télécommande, cette liste est automatiquement chargée dans le module. Au niveau de l'interface, le module a un aspect similaire à celui du lecteur de CD audio.

Figure 13-9
Module de lecture MP3.

Les informations supplémentaires concernant les caractéristiques techniques du fichier (fréquence d'échantillonnage, *bitrate*, mode stéréo/mono) ou bien les informations du type nom de l'artiste ou du morceau constituent les *ID3 tags* (marqueurs ID3) et sont extraites du fichier par le programme mpg123. On peut donc obtenir un affichage de ce type :

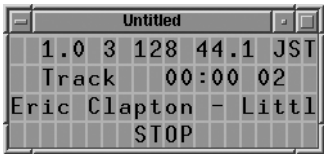


Figure 13-10
Affichage des marqueurs ID3.

Module d'encodage MP3

Ce module permet d'encoder des pistes audio extraites de CD musicaux en fichiers au format MP3. Le système OMM ne disposant pas de système d'encodage matériel, cette action sera exécutée en plusieurs phases :

- marquage des pistes à encoder comme indiqué précédemment,
- extraction des pistes depuis les CD concernés et sauvegarde sous forme non compressée (WAV) sur le disque,
- encodage au format MP3 puis suppression des fichiers non compressés.

Une phase intermédiaire avant l'encodage permet d'éditer la liste avant de procéder à l'encodage. Cette fonction est réalisée à l'aide du sélecteur de fichier et est accessible depuis la touche EDIT de la télécommande. L'extraction des pistes est réalisée grâce à l'utilitaire cdparanoia livré en standard dans les distributions Linux. À partir d'une interface ligne de commande, l'extraction est très simple.

```
# cdparanoia 1 piste_1.wav
cdparanoia III release 9.8 (March 23, 2001)
(C) 2001 Monty <monty@xiph.org> and Xiphophorus

Report bugs to paranoia@xiph.org
http://www.xiph.org/paranoia/

Ripping from sector       0 (track  1 [0:00.00])
          to sector   19049 (track  1 [4:13.74])

outputting to piste_1.wav

 (== PROGRESS == [ >                    | 000948 00 ] == :-) . ==)
```

La version standard du programme affiche la progression sous forme d'une ligne de texte utilisant le caractère >. Le symbole :-) (*smiley*) informe l'utilisateur du bon déroulement de l'extraction. La modification apportée dans la version OMM est la possibilité de piloter le programme à distance et de récupérer la valeur de la progression sous la forme *@P pourcentage état* (exemple : "@P 51% :-)"). Cette modification est aisée car, contrairement à mpg123, cdparanoia ne nécessite pas de dialogue complexe avec le programme appelant au cours de son exécution.

L'encodeur commence par demander le nom de la compilation qui sera associée aux pistes audio couramment sélectionnées. La saisie des caractères est un problème pour ce type de produit car nous ne disposons pas d'un clavier réel. Le principe adopté ici est celui des téléphones mobiles, et la télécommande contient donc des touches numériques couplées à des lettres. Pour entrer une lettre, il suffit d'appuyer autant de fois que nécessaire sur la touche associée. Pour passer à la lettre suivante, il faut utiliser la flèche qui se trouve à droite. Pour simplifier la saisie, le système ne gère que des lettres majuscules.

Figure 13-11
Saisie du nom de compilation.

Lorsque le nom est défini, le système demande successivement les CD associés aux différentes pistes à extraire.

Figure 13-12
Insertion du CD.

L'extraction des pistes s'effectue au fur et à mesure et les fichiers correspondant aux pistes audio sont stockés sur le disque dur sous forme de fichiers au format WAV.

Figure 13-13
Extraction d'une piste.

Lorsque toutes les pistes sont extraites, les fichiers WAV sont compressés au format MP3. Pour ce faire, nous utilisons également un outil Open Source disponible sur Internet. Il existe divers outils de ce type comme lame, bladeenc ou bien gogo. Originaire du Japon, ce qui explique sa sonorité quelque peu détonnante aux oreilles occidentales, gogo est de loin le plus rapide étant écrit en bonne partie en langage assembleur x86. Les modifications apportées sont là aussi mineures et concernent le pilotage à distance et l'affichage de la progression. Dans ce cas-là aussi, il serait tout à fait possible de remplacer un programme d'encodage par un autre à condition que celui-ci respecte l'interface de pilotage définie par OMM.

```
#ifdef OMM
        percent=100*frameNum/total_frame;
        printf("@P %d%%\n", percent);
        fflush(stdout);
#else
```

L'affichage est similaire à celui de l'écran d'extraction.

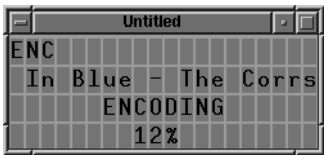


Figure 13-14
Encodage MP3.

Modules client NAPSTER

Le client NAPSTER est certainement le plus original du projet OMM. D'un point de vue conceptuel, NAPSTER vise à fournir aux utilisateurs une base de données qui indexe des milliers de fichiers musicaux MP3 répartis sur les postes des internautes du monde entier, ces derniers utilisant un programme « client » NAPSTER pour échanger ces fichiers. Bien que très novateur, le concept ne fut pas du goût des grosses sociétés d'éditions musicales et l'on peut aujourd'hui considérer qu'après des années de bataille juridique, NAPSTER est aujourd'hui bel et bien mort !

En nous en tenant au seul point de vue technique, il est sûr que les concepteurs de NAPSTER ont ouvert la voie à la notion de stockage réparti sur des postes clients à travers Internet. D'autres programmes similaires mais qui n'utilisent pas de base centralisée d'indexation (donc difficilement « repérables ») sont aujourd'hui disponibles, par exemple *Gnutella* (*http://www.gnutella.com*).

L'ajout d'un client NAPSTER à un produit de salon comme OMM est un défi, surtout en raison des limitations de l'affichage, les clients NAPSTER étant en général des applications graphiques évoluées. Sous Linux quelques produits cependant utilisent une interface de type texte et une fois encore le principal travail consiste à ajouter une interface de pilotage à distance. Le client NAPSTER est basé sur un programme Open Source appelé nap dans sa version 1.4.

La version OMM du client NAPSTER est également limitée à du téléchargement de fichier. Lorsqu'on clique sur la touche NAP de la télécommande, on obtient l'écran suivant, qui indique le nombre de fichiers disponibles ainsi que le nombre de bibliothèques.

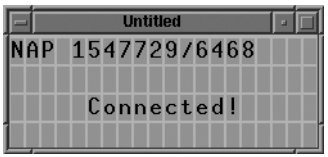


Figure 13-15
Client NAPSTER.

Ce type de client a pour principale fonction d'effectuer une recherche en fonction d'une chaîne de caractères. En appuyant sur la touche *Search* de télécommande, on obtient l'écran suivant :

Figure 13-16
Recherche de fichier.

À cause de la limitation géométrique de l'afficheur LCD, les résultats de la recherche sont indiqués un par un en indiquant la taille du fichier et le type de connexion.

Figure 13-17
Résultat de la recherche.

En téléchargeant le fichier, on obtient l'écran suivant :

Figure 13-18
Téléchargement du fichier.

On peut procéder au téléchargement en menant en parallèle une autre recherche. Comme dans un client NAPSTER classique, on peut revenir à l'écran de téléchargement en appuyant sur la touche *Downloads*. Les fichiers téléchargés sont ensuite utilisables de la même manière que ceux qui sont encodés à partir de pistes audio.

En résumé

OMM est un exemple typique d'utilisation de Linux pour une application embarquée communicante. L'utilisation exclusive de composants Open Source permet de réaliser facilement les fonctions recherchées en se basant sur des programmes existant en mode texte et en y ajoutant des couches de dialogue avec l'application (pilotage à distance à travers des tubes nommés). La simulation de l'environnement réel dans un poste de développement Linux permet de mettre au point bien plus aisément l'interface.

D''autres projets similaires sont aujourd'hui en cours, permettant également la lecture de fichiers vidéo de type MPEG4 et DivX. Nous pouvons citer entre autres :

- Le projet OM3 sur *http://www.enseirb.fr/~kadionik/om-cube/om-cube.html*
- Le projet GeeXxBox sur *http://www.geexbox.org/fr/screenshot.html*

14

Station Internet

Dans ce chapitre, nous allons présenter la seconde étude de cas de cet ouvrage. Il s'agit de réaliser une station Internet basée sur le navigateur web Netscape Navigator version 4.79. La station en question est constituée d'une unité centrale de faible encombrement basée sur un processeur Cyrix MediaGX compatible x86. Le système utilise un clavier à infrarouge intégrant également la fonction de pointeur (souris). La figure suivante donne l'aspect extérieur du matériel utilisé. Le système peut également mettre en œuvre un couple classique clavier/souris, tous deux connectés sur des ports PS/2.

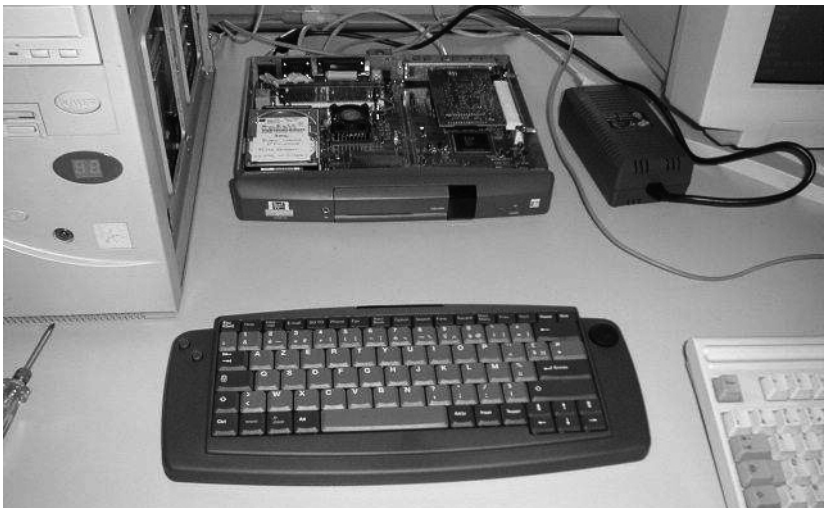


Figure 14-1

Aspect extérieur du système.

Le travail réalisé est basé sur les concepts décrits au chapitre 12 consacré aux interfaces graphiques. Le système peut être configuré à travers le navigateur pour les paramètres de connexion, ce qui nécessitera d'installer un serveur HTTP local. Le système utilise la version 3 de XFree86.

Le présent chapitre s'attachera à décrire les points suivants :

- l'intégration du navigateur sur le système,
- la gestion du clavier et de la souris infrarouge,
- l'intégration du serveur HTTP local (boa),
- la mise en place du système de configuration graphique.

Nous considérerons que le système est relié au réseau *via* une connexion Ethernet et ne traiterons donc pas le problème des connexions PPP. Ce dernier point est cependant traité en détails au chapitre 6.

Intégration du navigateur

Netscape Navigator est à ce jour le navigateur web le plus répandu sous Linux même si Opera (*http://www.opera.com*) ou Mozilla (*http://www.mozilla.org*) deviennent des compétiteurs très utilisés. Le choix du navigateur n'a en fait que peu d'importance sur la logique d'intégration sauf si nous en utilisons des particularités comme le pilotage à distance pour Navigator, décrit plus bas. Par rapport à Communicator, le terme Navigator indique la version programme qui intègre uniquement la fonction de navigation web (*navigator_standalone*) et non les fonctions de courrier électronique et forums de discussion intégrées par la version complète (*complete_install*). Le programme peut être téléchargé à l'adresse *ftp://ftp.netscape.com/pub/communicator.*

Le problème principal des navigateurs est l'espace mémoire utilisé tant au niveau du disque (ou de la mémoire flash) que de la mémoire vive. Même Opera, le plus léger des navigateurs « complets », occupe plusieurs méga-octets d'espace disque s'il intègre les extensions habituelles du langage HTML comme la gestion des *frames*, divers *plug-ins* ou, mieux encore, Java. Les utilisateurs désireux de ne pas utiliser de telles extensions, ce qui peut être le cas d'une application industrielle dédiée, pourront avantageusement s'orienter vers des programmes comme Dillo (*http://dillo.cipsga.org.br)*, un navigateur Open Source très léger (environ 200 Ko) et extrêmement rapide, déjà cité au chapitre 12.

Le fait est que l'archive du navigateur *stand-alone* occupe déjà 12 Mo en format compressé. Lorsqu'on extrait cette archive sur un répertoire temporaire, puis que l'on utilise le script `ns-install` pour installer le navigateur, on obtient un répertoire cible occupant environ 22 Mo.

```
# du -s /opt/netscape/
22612   /opt/netscape
```

Si l'on regarde ce répertoire de plus près, on peut décomposer le volume occupé assez facilement.

```
# cd /opt/netscape
# du -s *
20      LICENSE
324     Netscape.ad
20      README
8       XKeysymDB
8       bookmark.htm
6872    java
16      libnullplugin-dynMotif.so
408     nethelp
7568    netscape
5720    netscape-dynMotif
1616    plugins
4       registry
24      vreg
```

Si le plus gros élément est bien l'exécutable **`netscape`** lui-même (7,5 Mo), on peut déjà s'affranchir de la version **`netscape-dynMotif`** qui occupe 5,7 Mo. Cette version permet de gagner 2 Mo si le *toolkit* graphique Motif est installé sur le système sous forme de bibliothèques partagées. Comme ce n'est pas le cas, nous utilisons l'exécutable **`netscape`** correspondant à la version statique de Motif et nous pouvons donc supprimer la version dynamique. L'autre poids lourd est le répertoire **`java`** qui contient la machine virtuelle Java (JVM) fournie par Netscape. Si le support Java n'est pas nécessaire, on peut donc s'affranchir de 6,8 Mo de plus, mais il faudra invalider le support dans la configuration du navigateur. Le répertoire **`plugins`** contient en particulier le *plug-in* Flash permettant de visualiser les animations du même nom.

En résumé, une configuration avec Java occupe environ 16 Mo, la même sans Java environ 10 Mo. Ces valeurs sont cependant assez importantes et nous avons choisi d'utiliser pour la partition contenant le navigateur le système de fichier compressé Cramfs décrit au chapitre 7. À partir du répertoire **`/opt/netscape`**, on peut générer l'image au moyen de la commande :

```
# cd /opt
# mkcramfs . /archives/netscape.img
```

L'archive ainsi créée ne fait plus que 10 Mo au lieu de 16 Mo.

```
# ls -l /archives/netscape.img
-rw-r--r--    1 root     root     10825728 jun  3 16:57 /archives/netscape.img
```

Elle pourra être montée sur la cible par la commande :

```
# mount -t cramfs -o loop netscape.img /opt
```

et intégrée dans le fichier **`/etc/fstab`** de la cible par la ligne :

```
/archives/netscape.img      /opt          cramfs  loop    0 0
```

Le mode de démarrage dépend de l'utilisation du système. Si celui-ci n'intègre pas d'authentification par utilisateur (cas d'une borne interactive), l'interface graphique et le navigateur seront lancés au niveau du fichier **/etc/rc.d/rc.S** décrit au chapitre 5 :

```
# X GUI
HOME=/root
USER=root
cd $HOME
/usr/X11R6/bin/xinit -- -bpp 16

# Reboot if exits from Navigator
/sbin/reboot
```

La commande **xinit** permet de lancer le serveur X en démarrant la liste de clients décrite dans le fichier **.xinitrc** du répertoire d'accueil. Il faut noter que la sortie de l'environnement X provoque l'arrêt ou le redémarrage du système, ce qui est une bonne condition de sécurité. Dans notre cas, le fichier **.xinitrc** contient les lignes suivantes :

```
# WM
/usr/X11R6/bin/fvwm &
# Navigator
/opt/netscape/netscape
```

Le gestionnaire de fenêtre (ou *window-manager*) **fvwm** est très léger (environ 100 Ko) et s'adapte très bien à un environnement embarqué. On peut également imaginer d'utiliser le navigateur sans gestionnaire en utilisant un mode plein écran (appelé aussi « kiosque ») mais cela peut poser des problèmes en cas de création de nouvelles fenêtres par le navigateur et le mieux sera de mettre en place une gestion minimale des fenêtres, au moins pour gérer leur superposition (avant-plan/arrière-plan). La configuration du navigateur Netscape en mode plein écran se fait par l'intermédiaire du langage JavaScript en utilisant le code suivant :

```
<SCRIPT>
open('http://www.google.com', '_parent', 'location=no, menubar=no, resizable=no,
➥ scrollbars=yes, status=no, toolbar=no, outerHeight=480, outerWidth=640,
➥ screenx=0, screeny=0');
</SCRIPT>
```

Cela permet d'ouvrir l'URL *http://www.google.com* en mode plein écran 640 sur 480 pixels. Le mode plein écran est paramétrable par les arguments de la fonction JavaScript **open**. Si aucun élément de contrôle n'est disponible (cas décrit ci-avant), il est possible de piloter le navigateur à distance en utilisant l'option **-remote** de la commande **netscape** ou bien en utilisant le petit utilitaire **remote.c** disponible à l'adresse *http://wp.netscape.com/newsref/std/x-remote.html*. Pour provoquer l'affichage de la boîte de saisie d'une adresse, on fera :

```
netscape -remote 'openURL(http://home.netscape.com)'
```

Ce système permet de mettre en place assez facilement une interface personnalisée entièrement basée sur du code HTML. L'action sur des boutons provoque l'appel au

programme de pilotage à distance. Ce fonctionnement est cependant propre au navigateur Netscape.

Le langage JavaScript permet également de piloter une interface adaptée, et ce sur tous les navigateurs supportant ce langage. L'exemple présenté ci-après permet de gérer deux frames. Le frame supérieur (appelée *pub*) contient la partie active de la page qui est modifiée en fonction de l'action sur un bouton de la partie inférieure (appelée *menu*). La partie menu peut également être modifiée dynamiquement au cours de la navigation. Le code source du fichier racine **index.html** se résume à la création des deux frames.

```
<FRAMESET ROWS="640,500" FRAMEBORDER="1" BORDER="1" FRAMESPACING=0>
  <FRAME NAME="pub" SRC="pub1.html" SCROLLING=no>
  <FRAME NAME="menu" SRC="menu.html" SCROLLING=no>
</FRAMESET>
```

Le fichier **pub1.html** contient seulement le code de test suivant :

```
<BODY>
Contenu après action sur Menu1
</BODY>
```

Le fichier **menu.html** est une liste de pointeurs vers des pages provoquant la modification de la partie supérieure.

```
<BODY>
<TABLE>
<TR>
<TD><A HREF="menu1.html">Menu1</A></TD>
<TD><A HREF="menu2.html">Menu2</A></TD>
<TD><A HREF="menu3.html">Menu3</A></TD>
<TD><A HREF="web.html">Web</A></TD>
</TABLE>
</BODY>
```

Le fichier **menu1.html** contient uniquement le code JavaScript qui modifie les deux parties de l'écran (fonction **replace**). Les autres fichiers de menu sont similaires.

```
<SCRIPT>
parent.pub.location.replace ("pub1.html");
parent.menu.location.replace ("menu.html");
</SCRIPT>
```

Dans notre cas, la partie menu reste inchangée, sauf si l'on clique sur le pointeur d'accès au web, soit le fichier **web.html**.

```
<SCRIPT>
parent.www_win = open('http://localhost', '_blank', 'location=yes, menubar=no,
➥ resizable=no, scrollbars=yes, status=yes, toolbar=no, outerWidth=640,
➥ outerHeight=440, screenx=0, screeny=0');
parent.menu.location.replace ("web_menu.html");
</SCRIPT>
```

Dans ce fichier, une fenêtre d'accès au web est créée dynamiquement et est nommée **www_win**. Ce nom symbolique permettra ensuite de manipuler la page d'accès en utilisant d'autres fonctions JavaScript. La partie menu est maintenant modifiée en utilisant la page **web_menu.html** présentée ci-après.

```
<BODY>
<TABLE>
<TR>
<TD><A HREF="web_back.html">Back</A></TD>
<TD><A HREF="web_forward.html">Forward</A></TD>
<TD><A HREF="web_home.html">Home</A></TD>
<TD><A HREF="web_close.html">Close</A></TD>
</TR>
</TABLE>
</BODY>
```

Cette page présente des boutons de pilotage de la page web, soit *Back*, *Forward*, *Home* et *Close*, ce dernier permettant de fermer la page d'accès web et de revenir au menu local (fonction **close**). La page **web_back.html** est codée comme suit :

```
<SCRIPT>
parent.www_win.back();
parent.menu.location.replace ("web_menu.html");
</SCRIPT>
```

Les autres pages sont similaires, mais la page **web_close.html** retourne à l'écran local et pointe de nouveau sur le menu initial.

```
<SCRIPT>
parent.www_win.close();
parent.menu.location.replace ("menu.html");
</SCRIPT>
```

Ce petit exemple simple basé sur des commandes JavaScript élémentaires montre que l'on peut piloter un navigateur avec une interface très réduite et peu gourmande en ressources.

Gestion du clavier et de la souris infrarouge

Le système peut utiliser un clavier infrarouge combinant les fonctions de clavier et de pointeur de souris. Côté unité centrale, le récepteur est connecté à un port série qui manipule les trames du clavier sous forme d'une suite d'octets. Le but est de mettre en place un système permettant d'intégrer la gestion de ce clavier à l'interface X, et donc au navigateur. Une solution possible est d'écrire un module d'extension *Xinput* au serveur X. Cette extension permet d'ajouter dynamiquement des périphériques d'entrée au serveur. Elle est surtout utilisée pour gérer des tablettes graphiques et autres périphériques de CAO. L'ajout d'une extension se fait au niveau du fichier de configuration **XF86Config** situé sur **/etc/X11**. Les informations complètes concernant ce type d'extension sont disponibles sur le serveur *http://www.xfree86.org*. Pour la version 4 de XFree86, le terme

Xinput est remplacé par *inputDevice* et la syntaxe du fichier XF86Config est différente. Des informations intéressantes sont également disponibles sur la page personnelle de Frédéric Lepied à l'adresse *http://people.mandriva.com/~flepied/projects/wacom*.

Le problème principal que présente cette solution réside dans le fait qu'elle marche uniquement sous X et pas en mode texte, or il est intéressant de disposer d'un clavier (voire d'une souris) même en mode texte. On va donc traiter les données infrarouges dans un programme « démon » dédié, tournant en tâche de fond et lancé lors de l'initialisation du système. Le programme en question appelé **`ird`** (comme *Infra Red Daemon*) décode les trames clavier ou souris et les formate sous une forme compréhensible par le système.

Traitement de la souris

Le serveur XFree86 utilise le fichier de configuration **`/etc/X11/XF86Config`** dans lequel une section est réservée à la souris. La plupart du temps, cette section a l'apparence suivante :

```
Section "Pointer"
    Protocol    "PS/2"
    Device      "/dev/mouse"
EndSection
```

On fournit comme paramètres principaux le protocole utilisé par la souris (ici PS/2) et le device associé à cette dernière. Dans un système classique, le device **`/dev/mouse`** est souvent un lien symbolique **`/dev/psaux`** qui correspond au pilote de gestion d'une souris, connecté au port PS/2 du PC. Le plus simple est donc de fournir à XFree86 un flux de données correspondant à un protocole souris connu sur un device équivalent à **`/dev/mouse`**. XFree86 reconnaît un certain nombre de protocoles dont la liste est disponible dans la documentation du fichier **`XF86Config`** (**`man XF86Config`**). On peut citer entre autres :

- Logitech
- Microsoft
- MouseSystems
- PS/2
- etc.

Le protocole *MouseSystems* semble assez facile à utiliser ; il suffit de générer une trame de cinq octets à partir du numéro de bouton appuyé et des déplacements relatifs dx et dy du pointeur.

```
  char buffer[5];

  buffer[0] = (button ^ 0x07) | 0x80;
  buffer[3] =  dx - (buffer[1] = dx/2);
```

```
  buffer[4] = -dy - (buffer[2] = -dy/2);
  write (fd, buffer, 5);
```

Pour le canal d'envoi des données, on peut utiliser un tube nommé comme cela est décrit dans l'étude de cas précédente. On crée le tube au moyen du code suivant :

```
#define IR_FIFO   "/dev/irmouse"

  if (mkfifo (IR_FIFO,0666) && errno != EEXIST) {
    log_err ("mkfifo: %s", strerror(errno));
    Exit (1);
  }

  if ((fd_fifo = open (IR_FIFO, O_RDWR|O_NONBLOCK)) < 0) {
    log_err ("open: %s: %s", IR_FIFO, strerror(errno));
    Exit (1);
  }
```

On en déduit la nouvelle configuration du fichier **`XF86Config`**, très proche de la précédente :

```
  Protocol    "MouseSystems"
  Device      "/dev/irmouse"
```

Traitement du clavier

La gestion du clavier est plus compliquée même si elle est similaire. Selon l'état du système (en mode texte ou X), les données ne seront pas converties dans le même format et n'utiliseront pas la même technique d'insertion dans le système. Il est cependant aisé de connaître l'état du système en utilisant quelques fonctions bien choisies. Le code associé est en grande partie extrait du paquetage **`kbd`** qui fournit divers utilitaires de manipulation du clavier et de la console sous Linux. Nous rappelons que la configuration générale du clavier est décrite au début du chapitre 7.

À partir d'un descripteur de fichier **`fd_vt_stat`** associé à la console courante **`/dev/tty`**, on peut déterminer le type de console (texte ou graphique).

```
/* Determine la console virtuelle courante */
static int get_current_vt ()
{
  struct vt_stat vtstat;

  if (ioctl (fd_vt_stat, VT_GETSTATE, &vtstat)) {
    log_err ("get_current_vt: VT_GETSTATE: %s", strerror(errno));
    return -1;
  }

  return (vtstat.v_active);
}
```

```
/* Determine le type de console (texte/graphique) */
static int get_kd_mode ()
{
  char buf[256];
  int kdmode, current_vt;

  /* Teste si l'on est en mode console */
  if ((current_vt = get_current_vt ()) > 0) {
    sprintf (buf, "/dev/tty%d", current_vt);

    if ((fd_vt = open (buf, O_RDWR)) < 0) {
      log_err ("open: %s : %s", buf, strerror(errno));
      return -1;
    }

    if (ioctl (fd_vt, KDGETMODE, &kdmode)) {
      log_err ("get_kd_mode: KDGETMODE: %s", strerror(errno));
      return -1;
    }

    return kdmode;
  }

  return -1;
}
```

Si la console est de type texte, on peut simuler l'action sur la touche d'un vrai clavier en utilisant la commande **`TIOCSTI`** sur le descripteur de fichier **`fd_vt`** associé à la console courante.

```
ioctl (fd_vt, TIOCSTI, (char*)&c);
```

Le problème se complique cependant un peu lorsqu'on doit gérer les modificateurs de type *Control* ou *Alt* mais cela reste très accessible.

Si la console est de type X, nous utilisons l'extension XTEST du serveur X. Cette extension permet de simuler l'action sur le clavier en envoyant des événements X identiques. La présence de l'extension peut être testée de manière interactive grâce à la commande **`xdpyinfo`**.

```
$ xdpyinfo | grep XTEST
    XTEST
```

Le même test peut être effectué par programme.

```
if (!XQueryExtension (display, "XTEST", &major_opcode, &first_event, &first_error))

    fprintf (stderr, "XTEST extension not available.\n");
    XCloseDisplay (display);
    exit (1);
  }
```

L'envoi d'une chaîne de caractères par l'extension XTEST peut se faire en exécutant le code source suivant :

```
char *string;
Display *dpy;
unsigned long delay = 0;

KeySym ks = XstringToKeysym(string);
XTestFakeKeyEvent (dpy, XKeysymToKeycode (dpy, ks), 1, delay);
```

Une documentation complète concernant l'utilisation de XTEST est disponible en ligne à l'adresse *http://ftp.xfree86.org/pub/XFree86/4.0/doc/xtest.TXT*.

Système de configuration graphique

Dans les chapitres 5 et 6, nous avons expliqué comment la configuration du système était localisée dans des fichiers du répertoire **`/etc/sysconfig`** contenant des variables d'environnement. Ces valeurs sont ensuite utilisées par les scripts de démarrages du répertoire **`/etc/rc.d`**. Prenons ici l'exemple du fichier de configuration du réseau (fichier **`network`**), qui, rappelons-le, a la structure suivante :

```
# Network configuration
HOSTNAME=embtest
DOMAINNAME=localdomain
DEVICE=eth0
IPADDR=192.168.3.3
GATEWAY=192.168.3.1
NETMASK=255.255.255.0
NETWORK=192.168.3.0
BROADCAST=192.168.3.0
DNS1=62.161.120.17
DNS2=
DAEMONS="xinetd"
```

De la même manière, le fichier de configuration générale contient les lignes suivantes :

```
# General configuration
KEYTABLE=fr
KEYMAP=i386/azerty/fr-latin1
```

Ces fichiers peuvent bien sûr être aisément modifiés à l'aide d'un éditeur de texte mais il est plus convivial et plus sûr de donner la possibilité de modifier le paramétrage au moyen d'un navigateur web. Le résultat se présente de la manière suivante lorsqu'on ouvre la page de configuration sur le système.

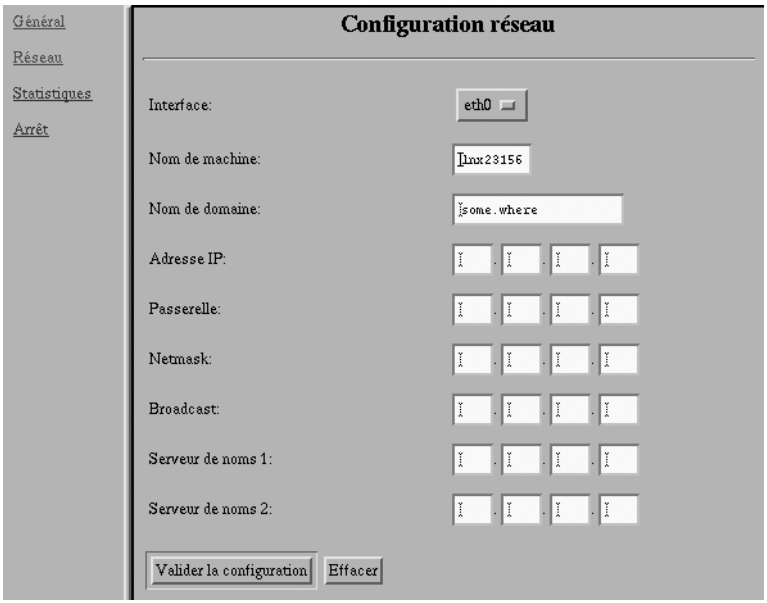


Figure 14-2
Configuration du système.

Une fonction de statistiques permet également de visualiser à distance l'état du système en présentant de manière plus élégante l'état des indicateurs extraits de `/proc`.

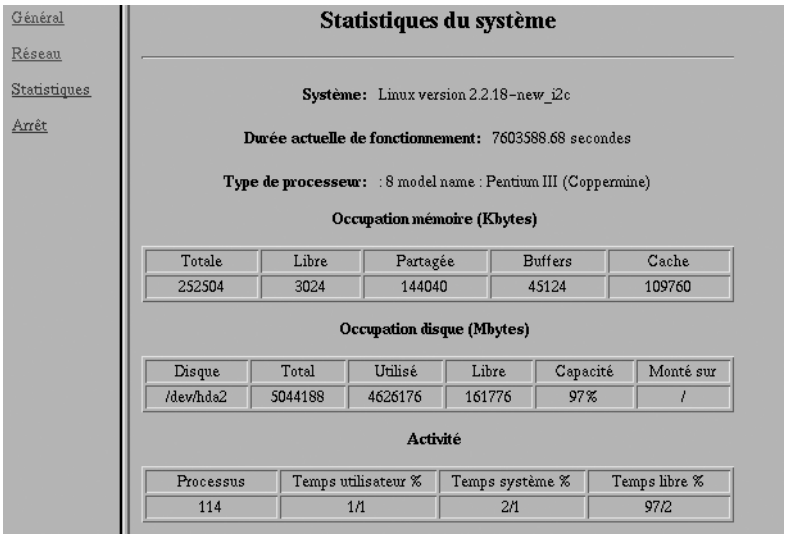


Figure 14-3
Statistiques du système.

Le paramétrage est basé sur l'utilisation de scripts CGI installés sur le système embarqué, ce qui nécessite bien sûr d'installer un serveur HTTP. Une méthode similaire a déjà été utilisée au chapitre 10.

La plupart du temps, les scripts CGI sont écrits en langage Perl , en langage Python ou en TCL. Au vu de la simplicité des scripts, et dans un souci permanent de réduction de l'espace utilisé, les scripts sont ici écrits en langage de script shell car les interpréteurs **/bin/bash** ou **/bin/ash** sont déjà installés sur le système. Concernant le serveur HTTP, nous n'utilisons pas le classique Apache (*http://www.apache.org*) livré en standard sous Linux mais le serveur réduit Boa (*http://www.boa.org*), largement suffisant dans notre cas et occupant seulement 50 Ko sur le disque.

L'installation du serveur Boa est relativement simple mais la documentation est assez succincte. L'archive des sources est récupérable à partir de l'adresse *http://www.boa.org*. Une fois l'archive extraite, on peut compiler le programme en exécutant :

```
$ cd src
$ ./configure
$ make
$ ls -l boa
-rwxr-xr-x    1 pierre   users      214130 Jun  4 18:03 boa
$ strip boa
$ ls -l boa
-rwxr-xr-x    1 pierre   users       56188 Jun  4 18:03 boa
```

Le programme peut être ensuite installé sur le répertoire **/sbin** de la cible ; il sera de ce fait lancé automatiquement au démarrage du système par le script **/etc/rc.d/rc.inet2** comme décrit au chapitre 6. Pour fonctionner correctement, Boa nécessite la présence du fichier **/etc/boa/boa.conf** qui est similaire à celui d'Apache bien que plus simple. Pour le bon fonctionnement du serveur, il convient de s'intéresser à quelques paramètres de ce fichier.

Le paramètre **Port** définit le port TCP utilisé par le serveur. La valeur par défaut est 80 mais, si l'on teste Boa sur un système utilisant déjà Apache sur le port 80, il faut impérativement modifier la valeur avant de lancer la commande **boa**.

```
# Port: The port Boa runs on.  The default port for http servers is 80.
# If it is less than 1024, the server must be started as root.

Port 80
```

Les valeurs **User** et **Group** qui définissent les identifiants et groupes lors de l'exécution du serveur sont en général fixées à *nobody*, valeur qui doit bien sûr exister comme utilisateur et groupe dans les fichiers **/etc/passwd** et **/etc/group** de la cible.

```
#  User: The name or UID the server should run as.
# Group: The group name or GID the server should run as.

User nobody
Group nobody
```

Remarque

Pour des raisons évidentes de sécurité, il est fortement déconseillé d'exécuter le serveur en tant que *root* surtout si l'on utilise des scripts CGI.

Afin de permettre l'utilisation de scripts CGI en dehors du répertoire dédié à ces scripts, il est nécessaire de valider l'option suivante :

```
# Uncomment the next line if you want .cgi files to execute from anywhere
AddType application/x-httpd-cgi cgi
```

Concernant les traces de fonctionnement du serveur (*log files*), le fait d'utiliser un système embarqué oblige à terme à limiter la taille des fichiers de trace, ou même à envoyer ces traces sur le fichier poubelle **`/dev/null`** lorsque la phase de mise au point est terminée.

```
# ErrorLog: The location of the error log file. If this does not start
# with /, it is considered relative to the server root.
# Set to /dev/null if you don't want errors logged.
# If unset, defaults to /dev/stderr

#ErrorLog /var/log/boa/error_log
ErrorLog /dev/null
```

Les documents HTML gérés par Boa doivent être placés par défaut sur le répertoire **`/var/www`**. Lorsque la configuration du fichier **`boa.conf`** est terminée, le serveur peut être testé en exécutant :

```
# /sbin/boa
```

Si un fichier **`/var/www/index.html`** existe et contient les lignes :

```
<BODY>
Hello world Boa!
</BODY>
```

l'accès à l'adresse *http://adresse_IP_du_systeme* à l'aide d'un navigateur doit afficher *Hello world Boa!*

Par défaut, les fichiers HTML et scripts CGI associés à la configuration du système sont installés sur **`/var/www/config`**. Le fichier **`index.html`** associé à ce répertoire est très simple, il correspond au résultat de la figure 14-2. Il utilise uniquement deux frames HTML, la première correspondant au menu et la seconde présentant l'écran d'édition des paramètres en fonction des choix dans ce menu. Cette fenêtre est associée à un script CGI qui calcule dynamiquement le code HTML en fonction des fichiers de paramètres du répertoire **`/etc/sysconfig`**. Le code de la page principale est le suivant :

```
<HEAD>
<TITLE>Configuration du système</TITLE>
</HEAD>

<FRAMESET  FRAMESPACING="0" COLS="100,*">
```

```
    <FRAME NORESIZE NAME="Menu" SCROLLING="auto" SRC="menu.html">
    <FRAME NAME="Main" SRC="network.cgi">
</FRAMESET>
```

Si l'on clique sur *Général*, la page de droite affiche la configuration générale du système qui se résume pour l'instant au choix du type de clavier. Nous allons détailler les scripts associés car ils sont beaucoup plus simples à appréhender que ceux de la configuration réseau, bien que la structure soit similaire. Le script **general.cgi** a pour but de lire le fichier de configuration **general** puis de présenter les informations du fichier dans la page HTML. Après modification éventuelle des paramètres par l'utilisateur, ces derniers seront sauvés sur le fichier de configuration par le script **set_general.cgi**.

Le début du fichier correspond à l'envoi de l'en-tête standard utilisé par les scripts CGI (les deux commandes **echo**). La ligne suivante permet d'évaluer les fichiers **variables** et **general** dans l'environnement du CGI. Dès lors, les valeurs des variables d'environnements définies dans ces fichiers sont disponibles au niveau du script CGI.

```
#!/bin/sh
echo Content-type: text/html
echo
. /etc/sysconfig/variables
. $BASE/$GENCFG
```

Nous rappelons que le fichier variable contient des lignes du type :

```
export BASE=/etc/sysconfig_test
export NETWORKCFG=network
export GENCFG=general
```

En fonction de la valeur de la variable **KEYTABLE**, on détermine le choix qui sera présenté à l'utilisateur dans la page HTML dynamique.

```
# Inits par défaut
if [ "$KEYTABLE" = "" ]; then
  KEYTABLE=fr
fi

# Lecture KEYTABLE
case "$KEYTABLE" in
fr) S_FR=SELECTED;;
us) S_US=SELECTED;;
*) S_FR=SELECTED;;
esac
```

La suite constitue la génération dynamique du code HTML qui est réalisé par un simple **cat** sur la sortie standard laquelle, dans ce cas, correspond à la connexion réseau avec le navigateur.

```
cat << EOF

<BODY>
<CENTER>
<H1>Configuration générale</H1>
</CENTER>
<HR>
<P>

<FORM ACTION="set_general.cgi">

<TABLE COLS=2>
<TR>
<TD WIDTH=$MAXWIDTH>Type de clavier:</TD>
<TD>
<SELECT NAME="KEYTABLE" VALUE="$KEYTABLE">
<OPTION $S_US>us
<OPTION $S_FR>fr
</SELECT>
</TD>
</TR>
</TABLE>

<P>

<INPUT NAME=VALIDER TYPE=submit VALUE="Valider la configuration">
<INPUT NAME=ANNULER TYPE=reset VALUE=Effacer>
</FORM>
</BODY>
EOF
```

La page utilise une forme HTML qui permet de saisir les nouveaux paramètres, puis de passer cette valeur sous forme de variable **KEYTABLE** au script CGI **set_general.cgi** associé à la forme lorsque l'utilisateur clique sur le bouton *Valider la configuration*. Cette technique a déjà été utilisée à la fin du chapitre 10 consacré à µClinux. La valeur de **KEYTABLE** est passée à ce script au moyen de la variable d'environnement **QUERY_STRING**.

Le script **set_general.cgi** doit extraire la liste des variables codée dans **QUERY_STRING**, et il utilise pour ce faire le code suivant basé sur la commande **sed** (*Stream EDitor)*.

```
#!/bin/sh

echo Content-type: text/html
echo

. /etc/sysconfig_test/variables
```

```
# Lecture des variables
if [ "$QUERY_STRING" != "" ]; then
    QUERY=`echo $QUERY_STRING | sed -e "s/=/='/g" -e "s/&/';/g" -e "s/+/ /g"
                                ➥ -e "s/$/'/"`
    eval $QUERY
```

Dès lors, les valeurs des variables passées par **QUERY_STRING** sont disponibles dans l'environnement local du script CGI car, si nous avons une chaîne du type :

```
X=1&Y=2&Z=3
```

dans la variable **QUERY_STRING**, le résultat de l'exécution de la commande **sed** donnera le résultat suivant dans la variable **QUERY** :

```
X='1';Y='2';Z='3'
```

Après évaluation par la commande **eval**, on obtient bien les valeurs de X, Y et Z dans l'environnement courant :

```
$ eval $QUERY
$ echo $X $Y $Z
1 2 3
```

La suite du script CGI se contente de calculer le nom réel du fichier de description du clavier à charger, de sauvegarder les valeurs sur le fichier **general** et d'envoyer un message de confirmation ou d'erreur à l'utilisateur.

```
case $KEYTABLE in
fr) KEYMAP=i386/azerty/fr-latin1;;
*) KEYMAP=i386/qwerty/us;;
esac

cat << EOF > $BASE/$GENCFG
# General configuration
KEYTABLE=$KEYTABLE
KEYMAP=$KEYMAP

EOF

ERR=$?

# Résultat
if [ $ERR != 0 ]; then
  echo "<BODY><H1>Erreur écriture config générale !</H1></BODY>"
  exit 1
fi

echo "<BODY><H1>Configuration enregistrée !</H1></BODY>
```

Pour que ce système fonctionne, le fichier `/etc/sysconfig/general` doit être en accès lecture/écriture pour l'utilisateur nobody utilisé par le serveur Boa. Il faut pour cela exécuter la commande `chown nobody:nobody` sur le fichier en question. Les erreurs d'accès sont tracées par défaut dans le fichier `/var/log/boa/error_log`.

En résumé

La construction d'une petite station simple de consultation basée sur Linux embarqué, X Window et un navigateur est relativement simple avec un peu de bon sens et en utilisant efficacement les composants du système. Ce petit exemple n'a pas la prétention de concurrencer une *set top box* professionnelle mais peut mettre sur la voie d'un projet réel plus ambitieux.

Glossaire

Glossaires en ligne

Il existe aujourd'hui un grand nombre de glossaires accessibles en ligne. Sur le sujet de l'embarqué, nous pouvons citer les liens suivants :

http://developers.cogentrts.com/cogent/cogentdocs/gl-timeglossary.html

http://www.redhat.com/embedded/technologies/glossary

http://www.rt.db.erau.edu/experiments/unix-rt/Glossary.html

ainsi que des liens plus généraux :

http://www.geocities.com/ikind_babel/babel/babel.html

http://www.zytrax.com/tech/electronics

http://www.cnet.com/Resources/Info/Glossary

Glossaire local

Ce petit glossaire non exhaustif reprend les définitions des principaux termes spécifiques utilisés dans cet ouvrage.

API

Acronyme anglais pour *Application Programming Interface* ou interface de programmation applicative. En général, cela signale un jeu de fonctions rassemblées dans une bibliothèque permettant de s'interfacer avec une couche logicielle donnée.

Appel système

Fonction accessible en mode utilisateur permettant d'appeler directement une fonction du noyau (exemple : `getpid()`, qui retourne la valeur du processus courant).

Appliance

Terme anglais se traduisant par « serveur dédié ». En fait, désigne un système dédié à une tâche donnée (serveur web, terminal Internet).

Boot

Démarrage d'un système informatique.

Changement de contexte

Actions accompagnant le passage d'un environnement d'un processus à un autre.

Collaboratif

Se dit d'un système multitâche dans lequel le noyau ne gère pas totalement le passage d'une tâche à une autre. Les applications doivent alors rendre la main au système pour assurer le changement de tâche.

Deadlock

En français, « étreinte fatale ». Situation de blocage sans limite de durée dans laquelle deux composants s'attendent mutuellement.

Déterminisme

Se dit d'un système dont le comportement génère les mêmes résultats à partir du moment où on lui applique les mêmes entrées.

Device

Terme général sous Unix qui désigne un fichier « banalisé » correspondant à une entrée dans le répertoire `/dev`. On peut parler en français de « périphérique » mais le terme est moins général.

Driver

Terme anglais pour « pilote de périphérique ».

Échéance temporelle

Délai d'exécution à respecter dans le cas d'un système temps réel à contraintes « dures » (*hard real time*).

Embedded

Terme anglais pour « embarqué ».

Enfoui

Terme français pour « embarqué ». Un système *deeply embedded* se traduit par « profondément enfoui ».

Flasher/flashage

Écriture de données sur une mémoire permanente de type mémoire *flash*. On parle également de *flasher* un système lorsqu'on met à jour le logiciel résidant sur une mémoire flash. Le terme n'est pas très élégant en français mais il est concis et efficace.

IPC

Acronyme anglais pour *Inter Process Communication* (communication inter-processus). Désigne un ensemble de composants logiciels permettant la communication entre des processus (mémoire partagée, sockets, messages).

JTAG

Acronyme anglais pour *Joint Test Advisory Group*. La norme JTAG ou IEEE 1149.1 permet de tester les parties numériques des systèmes, cartes électroniques et ASIC.

Kernel

Terme anglais pour « noyau », soit le cœur du système d'exploitation.

Logiciel embarqué

Logiciel dédié au fonctionnement d'un équipement matériel dont la fonction principale n'est pas forcément le traitement de l'information.

Mémoire partagée

Composant logiciel permettant de partager une zone de mémoire entre plusieurs processus (*voir* IPC).

MMU

Acronyme anglais pour *Memory Management Unit*. Composant matériel chargé de gérer la mémoire d'un système informatique.

Mutex

Composant logiciel qui permet de synchroniser des processus légers (threads) en définissant des sections critiques.

Ordonnanceur

Algorithme fondamental d'un système d'exploitation multitâche permettant le partage du temps entre les différents processus.

Posix

Acronyme anglais pour *Portable Operating System Interface*. C'est un ensemble de standards définissant le comportement de nombreux composants logiciels.

Predictible

Terme anglais. Se dit d'un système dont le comportement est prévisible. Proche du déterminisme.

Préemption

Au niveau du système d'exploitation, ce terme s'oppose à « collaboratif ». Dans un système multitâche préemptif, c'est le noyau qui décide d'interrompre une tâche pour passer à une autre. Au niveau d'un noyau, on parle de préemption lorsque ce dernier peut interrompre une tâche à n'importe quel moment afin de préserver l'ordre des priorités.

Priorité

Une valeur numérique qui définit l'ordre d'exécution d'une tâche par rapport aux autres.

Processus

Une tâche, souvent un ensemble de processus légers ou threads. Sous Unix, chaque processus est identifié par une valeur entière unique, appelée PID.

Quantum de temps

En anglais, *time slice*. Temps alloué à une tâche dans un système d'exploitation.

Ramdisk

En français, « disque mémoire ». Permet d'utiliser la mémoire vive comme une zone de stockage.

Root filesystem

En français, « système de fichier principal ». Correspond au premier système de fichier (appelé *slash* ou /) chargé par le noyau, et indispensable au fonctionnement du système.

RTOS

Sigle anglais pour *Real Time Operating System* (système d'exploitation temps réel).

Scheduler

Terme anglais pour ordonnanceur.

Semaphore

Composant logiciel permettant de gérer l'accès exclusif à une ressource.

Shell

En français, « coquille ». Sous Unix désigne l'interpréteur de commande `/bin/sh`.

Socket

Composant de communication inter-processus le plus souvent basé sur le protocole IP. Il existe cependant des sockets dites « locales » n'autorisant la communication qu'au sein d'un même système physique.

Temps partagé

Type de système d'exploitation pour lequel l'ordonnanceur privilégie le confort des utilisateurs au détriment du respect des échéances temporelles. Le comportement d'un tel système n'est pas déterministe et le système ne peut donc pas être utilisé pour des contraintes temps réel dures. Linux est un système de ce type. Les systèmes à temps partagé gèrent très mal les priorités entre les processus. On parle en anglais de *time sharing*.

Temps réel dur

À l'opposé du temps partagé, un système temps réel « dur » (*hard real time*) est basé sur la notion de priorité fixe. L'ordonnanceur a pour objet d'assurer le respect absolu des échéances temporelles.

Temps réel mou

Se dit d'un système qui doit répondre dans un temps raisonnable, « typiquement en X ms » ou « en moyenne en X ms ».

Thread

Terme anglais pour « processus léger ». Composant élémentaire d'un processus.

Tube

Traduction de *pipe*. Élément de communication inter-processus spécifique à Unix.

Index

Linux embarqué 2e édition

Pierre Ficheux

Ingénieur des Arts et Métiers, Pierre Ficheux est un linuxien de renom, initiateur de plusieurs sites Linux et auteur de nombreux tutoriels. Il a travaillé entre autres chez Red Hat et s'est spécialisé dans les applications industrielles de Linux embarqué. Pierre Ficheux est directeur technique d'Open Wide, société spin-off de deux grands groupes industriels français spécialisée dans les technologies Open Source.

Linux, solution idéale pour les systèmes embarqués

Discrets mais omniprésents, les logiciels embarqués équipent aussi bien les appareils électroménagers et les véhicules que les assistants personnels et les téléphones mobiles. Dans un contexte où robustesse, légèreté et interopérabilité sont essentielles, le système libre Linux se révèle un excellent choix : Open Source et libre de droits, il peut être adapté et diffusé à grande échelle pour un coût de licence nul, tout en intégrant l'ensemble des dialectes Internet et multimédias.

Un ouvrage de référence accompagné de deux études de cas

Sans équivalent en français, l'ouvrage de Pierre Ficheux commence par un panorama du marché de l'embarqué et des solutions Linux existantes en les comparant aux alternatives propriétaires. Il indique la méthodologie à suivre pour construire, à partir du noyau Linux, un système embarqué adapté. Cette nouvelle édition traite également de la prise en charge du noyau 2.6 ainsi que de l'utilisation et la création de chaîne de compilation croisée (ELDK et CROSSTOOL). Deux études de cas directement extraites de l'expérience industrielle de l'auteur décrivent la construction d'un lecteur/enregistreur CD/MP3 et d'une station de consultation Internet.

Au sommaire

À qui s'adresse cet ouvrage ?

– Aux développeurs Linux et aux ingénieurs ayant à réaliser des systèmes embarqués.
– Aux décideurs et industriels ayant à choisir une solution pour leurs applications embarquées.

@ Sur le site www.editions-eyrolles.com
– dialoguez avec l'auteur ;
– téléchargez le code source des études de cas ;
– consultez les mises à jour et compléments.

Code éditeur : G11674
ISBN : 2-212-11674-8

Conception : Nord Compo

EYROLLES