Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

Support de cours Réseaux EISTI

EISTI
Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

Table des matières

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

Présentation des Cours Réseaux

Organisation du cours

Le cours de réseaux est décomposé en deux parties, une partie cours magistral et une partie TD/TP.

Les cours magistraux auront lieu à l'EISTI. Ils porteront essentiellement sur la théorie des réseaux.

- Définition du terme réseau.
- Réseau *LAN/WAN* Différences et similitudes.
- Modèle en couches *OSI*, *ISO* et modèle *TCP/IP*.
- Principaux câblages :avantages / inconvénients.
- Présentation de divers *matériels actifs* réseaux.
- Notions d'*adressage* et de *routage TCP/IP*.
- Principes d'interconnexion de réseaux différents.

Les TP/TD auront lieu à l'EISTI. Pendant ces TD/TP, nous verrons la mise en œuvre de petites applications *Client/Serveur* utilisant *TCP/IP* en C sous *UNIX*.

Evaluation

L'évaluation sera réalisée suivant la catégorie 4

- examen final 70% de la note.
- travaux pratiques 30% de la note.

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

Cours 1 : Généralités

Que signifie réseau

Un réseau en général est le résultat de la connexion de plusieurs machines entre elles, afin que les utilisateurs et les applications qui fonctionnent sur ces dernières puissent échanger des informations.
Le terme réseau en fonction de son contexte peut désigner plusieurs choses. Il peut désigner l'ensemble des machines, ou l'infrastructure informatique d'une organisation avec les *protocoles* qui sont utilisés, ce qui 'est le cas lorsque l'on parle de Internet.
Le terme réseau peut également être utilisé pour décrire la façon dont les machines d'un site sont interconnectées. C'est le cas lorsque l'on dit que les machines d'un site (sur un réseau local) sont sur un réseau *Ethernet, Token Ring, réseau en étoile, réseau en bus,...*
Le terme réseau peut également être utilisé pour spécifier le *protocole* qui est utilisé pour que les machines communiquent. On peut parler de réseau *TCP/IP, NetBeui (protocole Microsoft) DecNet(protocole DEC), IPX/SPX,...*
Lorsque l'on parle de réseau, il faut bien comprendre le sens du mot.

Pourquoi des réseaux

Les réseaux sont nés d'un besoin d'échanger des informations de manière simple et rapide entre des machines. Lorsque l'on travaillait sur une même machine, toutes les informations nécessaires au travail étaient centralisées sur la même machine. Presque tous les utilisateurs et les programmes avaient accès à ces informations. Pour des raisons de coûts ou de performances, on est venu à multiplier le nombre de machines. Les informations devaient alors être dupliquées sur les différentes machines du même site. Cette duplication était plus ou moins facile et ne permettait pas toujours d'avoir des informations cohérentes sur les machines. On est donc arrivé à relier d'abord ces machines entre elles; ce fût l'apparition des réseaux locaux. Ces réseaux étaient souvent des réseaux "maisons" ou *propriétaires.* Plus tard on a éprouvé le besoin d'échanger des informations entre des sites distants. Les réseaux moyenne et longue distance commencèrent à voir le jour. Ces réseaux étaient souvent propriétaires. Aujourd'hui, les réseaux se retrouvent à l'échelle planétaire. Le besoin d'échange de l'information est en pleine évolution. Pour se rendre compte de ce problème il suffit de regarder comment fonctionnent des grandes sociétés. Comment pourrait-on réserver une place de train dans n'importe quelle gare? Sans échange informatique, ceci serait très difficile, voire impossible.

Pourquoi une normalisation

Si chacune des personnes (physiques ou morales) ne devait échanger des informations qu'avec des gens de sa communauté, alors il n'y aurait pas besoin de normalisation, chaque entité pourrait échanger ces informations avec des membres de la même entité. Il suffirait que chacune des personnes utilise le même "langage" (protocole) pour échanger ces informations.
Malheureusement (?), de plus en plus d'entité on besoin d'échanger des informations entre elles (SNCF, agence de voyage, organisme de recherche, école, militaires, ...). Si chacune de ces entités utilise son réseau (au sens protocole) pour que ces entités puissent communiquer ensemble il faudrait chaque fois réinventer des moyens pour échanger l'information. C'est ce qui se faisait au début. Des gens ont eu l'idée de réfléchir à ce problème et ont essaye de recenser les différents problèmes que l'on trouvait lorsque que l'on veut mettre des machines en réseau. De cette réflexion est sortie le modèle OSI de l'ISO.

Le modèle OSI de l'ISO

Pour faire circuler l'information sur un réseau on peut utiliser principalement deux stratégies.

L'information est envoyée de façon complète.

L'information est fragmentée en petits morceaux (*paquets*), chaque paquet est envoyé séparément sur le réseau, les paquets sont ensuite réassemblés sur la machine destinataire.
Dans la seconde stratégie on parle réseau à *commutations de paquets*.

La première stratégie n'est pas utilisée car les risques d'erreurs et les problèmes sous-jacents sont trop complexes à résoudre.

Le modèle OSI est un modèle à 7 couches qui décrit le fonctionnement d'un réseau à commutations de paquets. Chacune des couches de ce modèle représente une catégorie de problème que l'on rencontre dans un réseau.
Découper les problèmes en couche présente des avantages. Lorsque l'on met en place un réseau, il suffit de trouver une solution pour chacune des couches.
L'utilisation de couches permet également de changer de solution technique pour une couche sans pour autant être obligé de tout repenser.
Chaque couche garantit à la couche qui lui est supérieur que le travail qui lui a été confié a été réalisé sans erreur.

Couche	Fonctionnalité
7	Application
6	Présentation
5	Session
4	Transport
3	Réseau
2	Liaison
1	Matériel

La couche 1 Matériel

Dans cette couche, on va s'occuper des problèmes strictement matériels. (support physique pour le réseau). Pour le support, on doit également préciser toutes ces caractéristiques.

- Pour du câble :
  - Type (coaxial, paires torsadées,...)
  - si un blindage est nécessaire
  - le type du signal électrique envoyé (tension, intensité,...)
  - nature des signaux (carrés, sinusoïdaux,...)
  - limitations (longueur, nombre de *stations*,...)
  - ...
- Pour des communications hertziennes
  - Fréquences
  - Type de modulation (Phase, Amplitude,...)
  - ...
- Fibre optique
  - Couleur du laser
  - Section du câble
  - Nombre de brins
- ...

La couche 2 Liaison

Dans cette couche on cherche à savoir comment deux *stations* sur le même support physique (cf. couche 1) vont être identifiées. Pour ce faire, on peut par exemple assigner à chaque station une adresse (cas des réseaux Ethernet,....).

La couche 3 Réseau

Le rôle de cette couche est de trouver un chemin pour acheminer un paquet entre 2 machines qui ne sont pas sur le même support physique.

La couche 4 Transport

La couche transport doit normalement permettre à la machine source de communiquer directement avec la machine destinatrice. On parle de communication de bout en bout (*end to end).*

La couche 5 Session

Cette couche a pour rôle de transmettre cette fois les informations de programmes à programmes.

La couche 6 Présentation

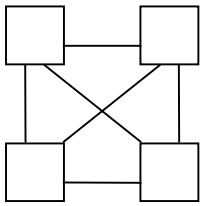
A ce niveau on doit se préoccuper de la manière dont les données sont échangées entre les applications.

La couche 7 Application

Dans la couche 7 on trouve normalement les applications qui communiquent ensemble. (Courrier électronique, transfert de fichiers,...)

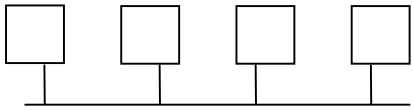
Architecture des réseaux

Câblage en maille



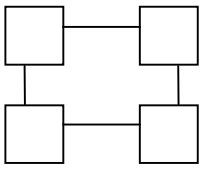
Chaque machine est reliée à toutes les autres par un câble.

Câblage en bus



Chaque machine est reliée à un câble appelé bus.

Câblage en anneau



Chaque machine est reliée à une autre de façon à former un anneau

Principes de fonctionnement

Maille:

Ce type de câblage n'est plus utilisé car il nécessite beaucoup de câbles.
Avec n machines il faut : n (n –1) /2 cables.

Bus:

Sur un câble de type bus, on utilise souvent un système CSMA/CD (Carriere Sense Multiple Acces / Collision Detection) Accès multiple avec détection de porteuse et détection des collisions.
Exemple : câblage *Ethernet*.
Lorsqu'une machine veut émettre un message sur le bus à destination d'une autre, la première commence par "écouter" le câble (CS). Si une porteuse est détectée, c'est que le bus est déjà utilisé. La machine attend donc la fin de la communication avant d'émettre ses données. Si le câble est libre, alors la machine émet ses données. Durant l'émission la machine reste à l'écoute du câble pour détecter une collision (CD). Si une collision est détectée, chaque machine qui émettait suspend immédiatement son émission et attend un délai aléatoire tiré entre 0 et une valeur N. Au bout du temps N le cycle recommence. Si une seconde détection est repérée le délai est tiré entre 0 et 2 * N. Ainsi de suite jusqu'à 16 * N. Après on recommence à N.
Chaque machine reçoit donc toutes les données qui circulent sur le bus. C'est au niveau de la couche 2 que l'on décide de garder les données ou de les jeter.

Anneau:

Les informations circulent toujours dans le même sens. Chaque machine qui reçoit un message, le recopie immédiatement sur le second câble. En même temps, l'information est remontée en couche 2 pour savoir si elle est doit être conservée par la machine ou détruite. L'information finira par revenir à la source. Cette dernière ne réemmettra pas l'information. Elle pourra comparer les données envoyées et les données reçus pour une éventuelle détection d'erreurs.

Sur un câble de type anneau on utilise souvent un système de *jeton*. Le jeton est un message particulier que les machines se font passer les une aux autres. Une machine n'a alors le droit d'émettre que lorsqu'elle dispose du jeton. Si la machine qui dispose du jeton n'a rien à émettre, alors elle fait passer le jeton à la

machine suivante. Il existe des algorithmes pour régénérer un jeton lorsque ce dernier est perdu suite à un incident.

Etoile:

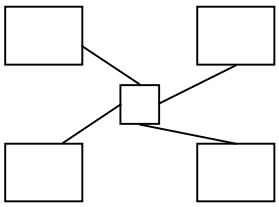
Sur un réseau en étoile toutes les communications passent par la machine qui est au centre de l'étoile. C'est cette dernière qui redirige l'information vers le destinataire.

Avantages et inconvénients

Le câblage en maile n'est plus utilisé car trop coûteux en câble.
De part son architecture, le câblage en bus avec des protocoles CSMA/CD convient très mal dans un environnement *temps réel.* Sur un réseau en bus, deux machines peuvent monopoliser le câble. L'architecture en anneau avec un protocole à base de jeton, peut servir dans un environnement temps réel car le délai maximum pour transmettre une information entre 2 machines peut être calculé. Le câblage en anneau nécessite plus de câble puisqu'il faut reboucler la dernière machine sur la première. Le câblage en anneau peut être perturbé par la panne d'une seule machine.
Dans une étoile, le point faible est le centre de l'étoile, si cet élément tombe en panne, alors tout le réseau est paralysé.

Techniques de câblage actuelles

De plus en plus on revient à un câblage qui ressemble à première vue à un câblage en étoile.



Chaque machine est reliée par un câble à un appareil *actif*. Ce type de câblage peut être utilisé dans une architecture réseau de type bus (Ethernet XXX BT). L'élément actif recopie alors l'information sur chacun des câbles. Dans une architecture de type anneau l'appareil réemet les informations sur un seul câble à la fois.

Cette architecture est plus sécurisée, car si une station tombe en panne (ou si son câble est défectueux), l'élément actif peut "désactiver" la ligne en défaut. Le seul rique reste au niveau du centre de l'étoile. Ce rique est limité, car le matériel est de plus en plus résistant.
Ce type de câblage est répendu car il permet d'utiliser les cables tirés par les téléphonistes.

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

Cours 2 : Couches 1 & 2

Etude des couches 1 et 2

Les couches 1 et 2 du modèle OSI sont souvent englobées dans l'adaptateur réseau.
Nous allons baser cette étude sur la technologie *Ethernet* et la technologie *Pronet-10.* La première est une topologie de type bus et la seconde une topologie de type anneau.
L'étude de ces 2 technologies du marché nous permettra de présenter 2 solutions aux problèmes des couches 1 et 2. Cette étude permettra de voir l'interaction entre les différentes couches et de fixer la notion d'adresses physiques.

Couche 1

Ethernet

Les réseaux Ethernet sont toujours très utilisés malgré l'âge de ce dernier. A l'origine seul le câblage en 10B5 existait. Aujourd'hui, on trouve de réseaux Ethernet en 10B2, 10BT ,100B2ou xxBF
Un nom de la forme xBy ce lit de la façon suivante: B : modulation de base; x bande passante du réseau (en méga bits par seconde)y définie le type du câble utilisé:

5 : câble coaxial de 1,7 cm de diamètre (gros Ethernet)
2 : câble coaxial de 0,5 cm de diamètre (Ethernet fin, cheapernet)
T: paires torsadées.
F: Fibre optique.

Câblage en 10B5

Ethernet est le nom que Xerox a donné à cette technologie, au cours des années 1970. Bien que "vieux" par rapport à l'évolution des systèmes informatiques, les réseaux locaux Ethernet sont toujours présents. Aujourd'hui encore, lorsqu'on envisage la création d'un réseau local, on pense souvent Ethernet. La version présentée ici est une version qui a été normalisée par les sociétés Intel, Xerox et DEC.
A l'origine un réseau Ethernet était matérialisé par un câble coaxial de couleur jaune d'environ 1,7 cm de diamètre. Sur ce câble, les machines ne peuvent être connectées que tous les multiples de 2,5m. Pour facilité les mesures, sur le câble normalisé de couleur jaune, on trouve une bague noire tous les 2,5m. La connexion d'une nouvelle machine (souvent appelée station) se fait via l'intermédiaire d'une prise "vampire". La pose de cette dernière ne nécessite pas de rupture du câble donc d'interruption du réseau. La prise est constituée d'une partie connectique, qui dérive une partie du signal électrique vers un dispositif électronique (appelé *Transceiver*). Le rôle du transceiver, est de détecter l'utilisation du câble et de transformer les signaux analogiques véhiculer sur le

câble en signaux numérique compréhensible par l'ordinateur. Chaque station est connectée à son transceiver par un câble 15 fils (appelé *Drop Câble*).

Voici quelques propriétés d'un câblage en 10B5:

- Chaque extrémité du câble est terminée par (un "bouchon") une résistance de 50 Ω entre l'âme et la tresse de blindage.
- La tresse de blindage doit être reliée à la terre à ces extrémités.
- La longueur maximale d'un segment est de 1500m.
- La longueur maximale du *drop câble* est de 100m
- Pour une courbure, l'angle maximal est de 120° sur un rayon minimum de 20 cm.

Câblage en 10B2

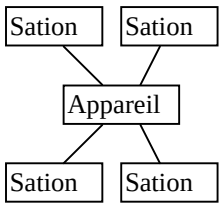
Le câblage en 10B2 plus connu sous les noms d'Ethernet fin", "thin Ethernet", "cheapernet " est une évolution récente du 10B5. Cette évolution due aux progrès de l'électronique permet de diminuer les coûts de câblage.
Les transceivers sont directement intégrés à la carte réseau. Sur chaque carte réseau on vient fixer un T disposant de connecteur BNC (2 femelles et un mâle que l'on branche sur la carte). Les stations sont reliées les unes aux autres par des cordons munis de connecteur BNC mâles que l'on connecte sur les T. Lorsque l'on veut insérer une station sur le bus, on est obligé de pratiquer une coupure sur le câble et de mettre des connecteurs BNC.

Voici quelques propriétés d'un câblage en 10B2:

- Chaque extrémité du câble est terminée par (un "bouchon") une résistance de 50 Ω entre l'âme et la tresse de blindage.
- La longueur maximale d'un segment est de 185m.
- La distance minimale entre 2 stations et de 50 cm.
- Le nombre de stations sur un segment est limité à 30

Câblage en 10BT ou 100BT

Cette technique de câblage a été prévue pour pouvoir utiliser les paires non utilisées par les gens des télécom dans les bâtiments. Il ne s'agit physiquement plus d'un câblage de type bus mais d'un câblage de type étoile. Toutes les stations sont connectées par des paires torsadées sur un élément actif (hub, switch,.)



Ce câblage de type étoile respecte également le principe CSMA/CD d'Ethernet puisque l'appareil (passif) rémét l'informations vers toutes les stations.
Il existe différents types d'appareils que nous allons détailler plus tard. Pour l'instant, nous allons supposer qu'à chaque fois qu'une station émet une information, l'appareil réemet cette information vers toutes les autres. Ainsi, on retrouve le principe de diffusion sur un bus.
Les limitations varient en fonction de la bande passante (10Mb ou 100Mb)que l'on souhaite obtenir.
Pour obtenir une bande passante de 100 Mb il faut que le câblage soit de catégorie 5. Ce qui implique des contraintes énormes sur la qualité du câble et sur la pose de ce dernier.
Pour obtenir une bande passante de 10 Mb il faut que le câblage soit de catégorie 3.
La catégorie 3 correspond en général au câblage utilisé par les téléphonistes.
La catégorie 5 nécessite une pose, et un câble, spécifiques.

Pronet-10

Le réseau Pronet-10 est un réseau de type anneau à jeton. Il est généralement câblé sur de la paire torsadée. Comme ce réseau est du type anneau à jeton, il est possible de garantir les temps de diffusion, il convient donc à un environnement temps réel pour peut que les protocoles des couches supérieures garantissent également les temps.
Sur ce réseau, on ne peut connecter au maximum que 254 machines (voir couche 2).

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

La couche 2

Maintenant que les machines sont reliées entre elles par un procédé physique, il reste à voir comment ces machines s'identifient pour échanger des informations sur le réseau (local). Pour ce faire, en général, chaque machine se voit attribuer une adresse physique, unique sur le réseau, qui permet de l'identifier. Plusieurs solutions sont possibles. En voici 2 exemples qui donnent une bonne idée des façons de procéder.

Pronet-10

Sur un réseau, l'administrateur assigne une adresse (comprise entre 1 et 254) à l'adaptateur grâce à de petits interrupteurs. Pour envoyer des informations chaque interface utilise des trames particulières:

Début de message 10 bits	Adresse Destination (8 bits)	Adresse Source (8 bits)	Typpe de trame (24 bits)	Données (0 à 16352 bits)	Fin de message (9 bits)	Parité (1 bit)	Refus (1 bits)
1	2	3	4	5	6	7	8

Les champs 1 et 6 contiennent une valeur conventionnelle qui sert simplement à indiquer que ce qui suit est bien des données et non pas du bruit, ou la fin du message.
Les champs 2 et 3 contiennent les adresses du destinataire et de la source du message.
Le champ 4 indique le type du message avec des valeurs conventionnelles (données, jeton,...)
Le champ 5 de longueur variable contient les données proprement dites avec un maximum de 2044 octets.
Le champ 5 sert à faire une vérification minimale.
Le champ 6 peut être positionné par le récepteur pour indiquer le refus de la trame.

Ethernet

Sur ce type de réseau, les adresses physiques sont attribuées directement par le constructeur de la carte. Elle est implémentée directement dans le matériel. Ces adresses sont codées en dur sur 48 bits (ce qui permet de connecter au maximum $2,8*10^{14}$ machines !!!).Ce système permet donc de connecter plus de machines (malgré les limitations données en couche 1) que sur un réseau Pronet-10. Le risque d'avoir 2 adresses physiques identiques sur le réseau est donc nul. Les trames Ethernet diffèrent légèrement des trames Pronet-10.

Préambule (64 bits)	Adresse du destinatiare (48 bits)	Adresse de la source (48 bits)	Typpe de trame (16 bits)	Données (368 à 1200 bits)	CRC (32 bits)
1	2	3	4	5	6

Champ 1, 2,3 ,4 ,5 voir trame Pronet-10
Champ 6: Champ pour contrôler la validité de la trame.

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

Améliorations d'Ethernet

Amélioration en nombre de stations

Sur un réseau Ethernet, en fonction du câblage utilisé, il existe des limitations soit en nombre de machines et/ou en longueur de câble. Sur un réseau local, on peut cependant dépasser ces limitations grâce à du matériel. L'ajout de ce matériel (*actif* ou *passif*) ne modifie pas les principes généraux. En particulier, lorsque l'on parlera d'interconnexion de réseaux, ce matériel sera complètement transparent.

Eléments passifs

Ce type de matériel intervient directement au niveau de la couche 1. Il prend le signal et l'amplifie.
On trouve des *répéteurs* pour les câblages en 10B5 et 10B2. On ne peut mettre que 2 répéteurs au maximum sur un réseau de type Ethernet.
Sur un câblage 10BT ou 100BT les appareils au centre de l'étoile peuvent être de type passif ou actif .

Eléments actifs

Ce type de matériel est dit *actif*, car il doit connaître le type des trames envoyées. Ces appareils sont considérés comme une station sur le bus, ils reçoivent des trames et les réemettent sur le second câble si ces dernières sont valides. On peut trouver des *ponts* (*bridge*), *multiports*, ... en 10B5 et 10B2.
En 10 BT, ce matériel n'existe pas, car il suffit d'interconnecter les hubs les uns aux autres.

Amélioration des performances

Le problème d'un réseau Ethernet est qu'à un instant donné, seulement 2 machines (sauf en diffusion) peuvent communiquer ensemble.
Il existe des appareils actifs qui vont permettre de segmenter le réseau physique en petit morceau pour du 10Bx. On trouve des variantes de ponts et de multiports qui sont dits filtrant. Ils agissent au niveau de la couche 2. En regardant l'adresse de l'émetteur et celle du destinataire (contenues dans la trame) l'appareil peut savoir s'il doit recopier ou non l'information sur les autres câbles.
Le principe en 10BT est différent, car les machines sont sur des câbles différents. L'idée consiste à "ne relier" à un moment donner (durant le passage de la trame) que les câbles des machines concernées. Si plusieurs couples de machines communiquent, l'appareil (un *switch*) établie plusieurs canaux de communication.

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

Cours 3 : Couche 3

Présentation

Le rôle de la couche 3 est de trouver un chemin pour faire communiquer 2 machines qui sont situées sur des réseaux différents interconnectés.
Ils existent plusieurs protocoles de couche 3 normalisés. Cependant ces derniers ne sont pas très utilisés, nous allons donc continuer l'étude sur TCP/IP qui ne suit pas le modèle OSI mais qui est très répandu.

TCP/IP présentation

TCP/IP est né de la réflexion de chercheurs américains suite à un problème posé par l'armée américaine. L'armée américaine disposait (et dispose encore) de plusieurs bases sur le territoire. Chacune de ces bases dispose de sa propre logistique informatique. Les machines des différents centres pouvaient être de types différents et reliées entre elles à l'intérieur de ces centres par des réseaux locaux différents. Cependant ces centres informatiques doivent échanger des informations. Les bases sont reliées les unes aux autres par des câbles. La question était de trouver un moyen pour que l'information puisse circuler entre ces bases même si certains des chemins empruntables étaient détruits. Il fallut donc trouver un système permettant de retrouver des chemins (*routes)* qui se reconfigureraient automatiquement en cas de coupures des liaisons.
De cette recherche est née IP (*Internet Protocol* ou Interconnected Network Protocol). IP comme nous le verrons, est un protocole qui permet d'envoyer des informations élémentaires de machine à machine. Cependant l'information ne part pas d'une machine mais d'une **application** fonctionnant sur une machine pour aboutir à une **application** fonctionnant sur une machine. Pour résoudre ce problème les chercheurs ont développé un autre protocole de nom *TCP (Transport Control Protocol).*
Le nom de TCP/IP a donc été choisi en référence à ces deux principaux protocoles qui le caractérisent.
Aujourd'hui TCP/IP intègre beaucoup d'autres protocoles (ICMP, IGP, FTP, SMTP, HTTP, ...).
TCP/IP est un protocole qui nécessite une coopération des *OS* des machines dans pratiquement toutes les couches. Dans un réseau qui suit le modèle OSI, l *OS* (Operating System : système d'exploitaion) de la machine n'intervient que dans les couches 4 et supérieures.
TCP/IP est très répandu, car sa robustesse a été prouvée (quelques millions de machines interconnectées dans le monde). Il est également très répandu, car dès son origine il a été implémenté sur des systèmes Unix. Beaucoup de chercheurs ayant contribué à l'évolution de TCP/IP à son origine sont issus de l'université de Berkeley qui a très largement diffusé son système Unix avec l'interface des ***sockets*** pour manipuler des connexions TCP/IP.

Vue en couches de TCP/IP

TCP/IP ne suit pas directement le modèle OSI parce que la normalisation OSI lui est postérieure. Cependant cette famille de protocole suit également un schéma en couche.

Application	
Transport	TCP, UDP
Interconnexion	IP
Interface avec le réseau	
Matériel	

La couche Matérielle correspond aux couches 1 et 2 du modèle OSI.
Les couches matérielles et Interface avec le réseau correspondent à la couche 3 du modèle OSI.
La couche Transport correspond aux couches 4 et 5 du modèle OSI.

Cette comparaison au modèle OSI n'est que relative, car chaque couche du modèle OSI doit vérifier que la couche équivalente sur la machine destinataire va recevoir **toutes** les données émises **sans erreur.** Le protocole des couches Interface avec le réseau et Interconnexion ne garantit pas ceci. Ces protocoles sont de type *Best Effort*. Le problème de traitement des erreurs est remonté dans les couches supérieures (Couche transport en utilisant TCP ou couche application en utilisant UDP).

Identification des machines

Sur un réseau utilisant TCP/IP chaque machine est identifiée par une adresse IP. Chaque identifiant IP appelé numéro ou adresse IP doit être unique sur l'ensemble du réseau. Chaque machine ne dispose que d'une adresse IP par réseau sur lequel elle est connectée. Les machines (routeurs, passerelles)qui sont *multi-domiciliées* c'est-à-dire qui possèdent plusieurs adresses IP sont des cas spéciaux que nous étudierons plus tard.

Format d'une adresse IP

Une adresse IP est un nombre codé sur 4 octets. Par habitude, cette adresse est représentée sous la forme décimale pointée w.x.y.z où w,x,y,z sont quatre chiffres décimaux allant de 0 à 255. Cette adresse peut être vue de 2 façons différentes:

La machine d'adresse w.x.y.z .
La machine d'adresse z du réseau w.x.y.0 .
La machine d'adresse y.z du réseau w.x.0.0 .

La machine d'adresse x.y.z du réseau w.0.0.0 .

Ces différentes façons de lire une adresse IP permettent d'optimiser la façon de calculer les routes (*routing, ou routage ???)*. La décomposition d'une adresse IP en adresse de réseau plus une adresse de machine sur un réseau ne se fait pas au hasard.

Les différentes classes d'adresses.

Pour voir si l'adresse du réseau d'une machine est codée sur 1,2 ou 3 octets, il suffit de regarder la valeur du premier. La valeur de l'octet x permet également de distinguer la **classe** du réseau.

Classe	Valeur de w	lg adresse réseau	Nb de réseau	nb max de machines
A	de 0 à 127	1 octet	127	16777216
B	de 128 à 191	2 octets	16384	65536
C	de 192 à 223	3 octets	2097152	256
D	de 224 à 239			
E	de 240 à 255			

Pour l'instant que la machine ait une adresse de classe A,B,C ne change rien au raisonnement que nous allons tenir. Ceci interviendra que lorsque nous verrons les problèmes de routage. Cependant il faut noter que dans une adresse IP, une partie de cette dernière sert également à identifier le réseau.

La classe E est réservée pour des extensions futures.

La classe D est la classe de diffusion de groupe. L'étude de ces adresses ne sera pas faite durant ce cours.

Dans cette partie du cours, nous supposerons que les machines qui échangent des informations via TCP/IP sont toutes situées sur le même réseau physique (éventuellement prolongé via des appareils actifs qui ne travail qu'en couche 1 et 2 du modèle OSI).

Passage des adresses IP aux adresses physiques.

Dans un réseau TCP/IP, nous avons dit que chaque machine était identifiée par une adresse IP. Cette adresse est logique, elle ne dépend pas du matériel utilisé pour relier les machines ensemble. Ces adresses IP peuvent être modifiées relativement rapidement par les administrateurs pour diverses raisons. Nous avons vu jusqu'à présent (couche 2 du modèle OSI) que chaque machine disposait d'une adresse **physique** différente. Cette adresse physique dépend du matériel réseau utilisé. Il faut trouver un système qui permette de convertir l'adresse logique IP en une adresse physique de la machine. Pour ce faire plusieurs méthodes sont utilisables

La table

On peut imaginer que sur chaque machine travaillant avec TCP/IP on dispose d'une table qui fait la conversion entre une adresse logique IP et une adresse matérielle type Pronet, Ethernet, ou ... . Cette méthode, quoi que très efficace, devient lourde à gérer. A chaque ajout, suppression ou modification d'une adresse IP pour une machine, il faut remettre à jour la table de correspondance sur toutes les machines.

La conversion directe

Avec des réseaux physiques dont les adresses doivent être paramétrées par l'administrateur, on peut supposer que ce dernier peut faire coïncider tout ou partie de l'adresse physique à l'adresse IP. Cette technique est très facile à mettre en œuvre sur un réseau Pronet, on peut par exemple décider que le dernier octet de l'adresse IP sera égal à l'adresse physique. Cette méthode ne peut cependant pas toujours être mise en œuvre (c'est le cas avec Ethernet).

La conversion dynamique (ARP)

Cette méthode de résolution d'adresses physiques est basée sur le principe suivant : chaque machine connaît son adresse IP et son adresse physique. Il faut donc trouver le moyen de demander à une machine dont on ne connaît que l'adresse IP de bien vouloir nous donner son adresse physique pour que l'on puisse lui envoyer les informations.

A première vue nous retombons sur le même problème : obtenir une adresse physique pour demander cette adresse physique.

Pour résoudre ce problème il faut que le réseau (couche 2) supporte la **diffusion** c'est à dire qu'il existe une "adresse physique" qui corresponde à toutes les machines.

Pour obtenir l'information, la machine qui veut émettre une information sur une machine distante va regarder si elle connaît l'adresse physique du destinataire. Si oui elle va directement lui envoyer cette information.

Sinon, elle va émettre en diffusion sur le réseau une **demande de résolution d'adresse**. Toutes les stations du réseau vont donc recevoir cette information. Dans cette demande, on trouve l'adresse IP dont on veut connaître l'adresse physique. La machine qui a l'adresse IP correspondante pourra envoyer une réponse contenant son adresse physique.

La correspondance Adresse physique / adresse IP sera gardée par la machine émettrice pendant un certain temps, de façon à ne pas reposer la question trop souvent. Cette information doit expirer au bout d'un moment, car la carte d'interface réseau du destinataire peut être changée donc probablement son adresse physique (c'est le cas avec Ethernet). Ce mécanisme est connu sous le nom d'**ARP** *(Adresse Resolution Protocol)*. ARP peut être utilisé avec tous types de réseaux supportant la diffusion. Il peut également être utilisé par n'importe quelles familles de protocoles en particulier avec TCP/IP.

La résolution inverse (RARP)

Connaître l'adresse physique d'une machine connaissant son adresse IP, permet de communiquer. Il y a cependant des cas où la machine ne connaît que sa propre adresse physique et souhaite obtenir son adresse IP.
Prenons le cas d'une machine qui démarre. Si cette machine démarre sur un disque, elle peut aller lire des fichiers de configurations et donc trouver son adresse IP. Dans ce cas, cette machine n'a pas de problème.
Si cette machine va chercher son OS sur le réseau, au démarrage elle ne connaît que son adresse physique. Pour obtenir un fichier image de son boot, elle doit utiliser des protocoles de transfert de fichiers qui sont souvent basés sur TCP/IP. Cette machine doit donc travailler avec TCP/IP et par conséquent connaître son adresse IP. Pour connaître son adresse IP en ne connaissant que son adresse physique, la machine peut utiliser **RARP***(Reverse Addresse Resolution Protocol)*./
Le principe est le suivant.
Sur le réseau, on doit avoir une ou plusieurs machines (**serveur RARP)** contenant des tables (mises à jour à la main) associant des adresses physiques à des adresses IP. La machine qui veut connaître son adresse IP envoie en diffusion sur le réseau une demande RARP. Les machines serveurs RARP vont donc recevoir cette demande et pouvoir donner l'adresse à la machine.
Cette dernière peut ainsi demander une image de son OS qui pourra être transférée avec des protocoles de hauts niveaux (tftp, bootp,...).

Cours 4 : Routage

Introduction

Nous allons étudier comment la couche "INTERCONNECTION" délivre des datagrammes (ou informations) à travers des réseaux interconnectés.
Nous aborderons la fragmentation de paquets et ces conséquences.

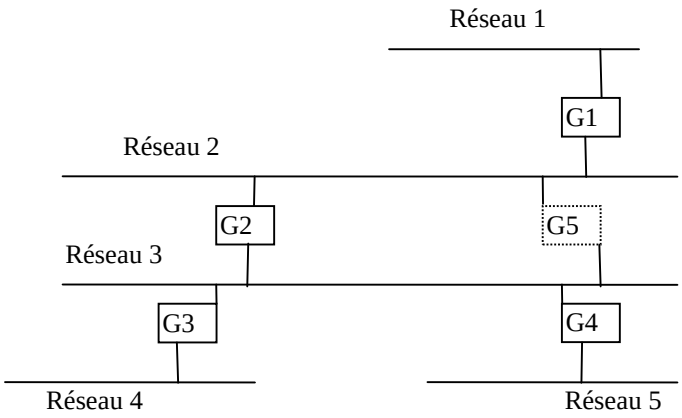
Principe d'un algorithme de routage

Un des protocoles les plus connus est RIP (RIP: ROUTING INFORMATION PROTOCOL) également connu sous le nom d'un programme qui le met en œuvre : *routed* . Le logiciel *routed* a été réalisé à l'université de Californie : Berkeley. Il assure un routage cohérent et permet l'échange d'informations d'accessibilité entre machines sur les réseaux locaux de cette université. Il Utilise la diffusion sur le réseau physique pour propager rapidement les informations de routage. Il n'a pas été initialement conçu pour être utilisé sur les réseaux grande distance. *Routed* s'appuie sur des travaux de recherche antérieurs menés par le centre de recherche de la compagnie Xerox *(PARC: Palo Alto Research Center), à Palo* Alto. Il met en œuvre un protocole dérivé du protocole d'informations de routage de Xerox (NS) et l'a généralisé à un ensemble de familles de réseau.

La popularité de *RIP*, en dépit de légères améliorations par rapport à ses précurseurs, ne tient pas à ses mérites techniques. Au contraire, elle résulte de la distribution de ce protocole avec les logiciels du célèbre système d'exploitation UNIX 4 BSD Ainsi, de nombreux sites TCP/IP ont adopté et commencé à utiliser *routed* et RIP, sans en avoir étudié les caractéristiques techniques ni les limitations. Une fois installé et opérationnel, il a constitué la base du routage local et les groupes de recherches l'ont adopté pour des réseaux plus grands. Le fait le plus surprenant concernant RIP est peut-être d'avoir été mis en œuvre bien avant que la norme (RFC plus exactement) correspondante ne soit spécifiée. La plupart des mises en œuvre sont dérivées du code de Berkeley et l'interfonctionnement de ce protocole est limité par la compréhension des détails et des subtilités non commentés qu'en avaient les programmeurs. Avec l'apparition de nouvelles versions, de nouveaux problèmes se sont posés un RFC est enfin apparue en juin 1988.

Le protocole RIP sous-jacent est une application directe du routage à vecteurs de distance utilisé pour les réseaux locaux. Il classe les participants en machines *passives ou actives.* Une passerelle active propage les routes qu'elle connaît vers les autres machines; les machines passives écoutent les passerelles et mettent à jour leurs routes en fonction des informations reçues, mais n'en diffusent pas elles-mêmes. Habituellement, les passerelles utilisent RIP en mode actif et les machines l'utilisent en mode passif.

Une passerelle utilisant RIP en mode actif diffuse un message toutes les 30 secondes. Le message contient des informations extraites de ses tables de routage courantes de la passerelle. Chaque message est constitue d'une partie contenant l'adresse IP d'un réseau et un entier mesurant la distance de la passerelle vers ce réseau. RIP utilise une métrique à nombre de sauts (hop *count metric)* pour mesurer la distance qui la sépare d'une destination. Dans la métrique RIP, une passerelle est située à une distance d'un saut d'un réseau directement accessible, a deux sauts des réseaux accessibles par l'intermédiaire d'une autre passerelle et ainsi de suite. Le nombre de sauts *(number of HOP OU hop count metric)* mesure donc le nombre de passerelles que doit traverser un datagramme pour atteindre sa destination, sur un chemin reliant une source donnée à une destination donnée. Il doit être clair que l'utilisation du nombre de sauts pour déterminer les plus courts chemins ne conduit pas toujours à une solution optimale. Par exemple, un chemin qui traverse trois réseaux Ethernet et comporte trois sauts peut être nettement plus rapide qu'un chemin qui ne comporte que deux sauts, mais emprunte deux liaisons séries basse vitesse. Pour compenser les différences inhérentes à la technologie, de nombreuses mises en œuvre de RIP utilisent des nombres de sauts artificiellement élevés lorsqu'elles diffusent des informations relatives des réseaux à bas débit.
Partenaires RIP actifs et passifs écoutent tous les messages et mettent à jour leurs tables de routage, conformément à l'algorithme de routage à vecteurs distance décrit précédemment. Ainsi, dans l'interconnexion présentée

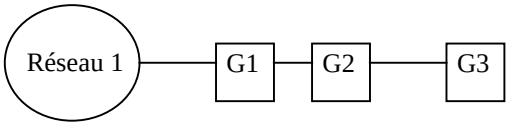


La passerelle Gl diffusera sur le réseau un message qui contient la paire (1, 1), ce qui signifie que G1 peut atteindre le réseau 1 pour un coût de 1. Les passerelles G2 et G5 reçoivent ces informations et mettent leurs tables à jour en créant une route qui paisse par la passerelle G1 pour atteindre le réseau 1 (pour un coût de 2). Ultérieurement, les passerelles G2 et G5 propagent la paire (1, 2) lorsqu'elles diffusent leur message RIP sur le réseau 3. Le cas échéant, toutes les passerelles et les machines créeront une route vers le réseau 1.

RIP définit quelques règles d'amélioration des performances et de la fiabilité. Ainsi, lorsqu'une passerelle apprend d'une autre passerelle l'existence d'une nouvelle route, elle conserve cette dernière jusqu'à ce qu'elle en connaisse une meilleure. Dans notre exemple, si les passerelles G2 et G5 propagent des informations de routage indiquant que le réseau 1 a un coût de 1, les passerelles G3 et G4 créeront une route passant par la première passerelle dont elles auront reçu le message. Nous pouvons le résumer ainsi:

Pour empêcher l'oscillation entre deux ou plusieurs passerelles de même coût RIP indique que les routes crées doivent être conservées jusqu'à ce qu'une route de coût strictement inférieur apparaisse.

Que se passe-t-il lorsque la passerelle qui a propagé les informations de routage tombe en panne ? RIP indique que tous les récepteurs doivent associer une temporisation aux routes acquises. Lorsqu'une passerelle définit une route dans ses tables, elle lui associe une temporisation et l'active. La temporisation doit être réarmée à chaque fois que la passerelle reçoit un message RIP référençant cette route. La route est invalidée s'il s'écoule 180 secondes sans qu'elle soit de nouveau référencée.
RIP doit prendre en compte trois types d'erreurs causés par l'algorithme sous jacent. D'abord, l'algorithme ne détecte pas les boucles de façon explicite, RIP doit donc supposer que ses partenaires sont fiables ou prendre des précautions pour détecter les boucles. RIP doit ensuite utiliser une petite valeur comme distance maximale possible (RIP utilise la valeur 16) pour éviter les instabilités. Les administrateurs de réseaux doivent utiliser un autre protocole pour les interconnexions dans lesquelles le nombre de sauts avoisine normalement la valeur 16 (de toute évidence, la petitesse de la valeur limite du nombre de sauts de RIP le rend inutilisable dans les grands réseaux).
Troisièmement, l'algorithme de routage à vecteurs de distance utilisé par RIP entraîne une convergence lente (slow *convergence) ou* un problème de valeur infinie (coun*t to infinity)* qui produit des incohérences parce que les messages se propagent lentement à travers le réseau. Le, choix d'une faible valeur infinie (16) limite le phénomène de valeur infinie, sans pour autant l'éviter.
L'incohérence des tables de routage n'est pas spécifique de RIP. C'est un problème fondamental inhérent à tout protocole à vecteurs de distance où les messages ne véhiculent que des paires (réseau destination, distance).
Considérons, pour comprendre le problème, l'ensemble de passerelles de la figure suivante :



La figure représente les routes vers le réseau 1 pour l'interconnexion le la figure précédente.
La passerelle Gl accède directement au réseau 1. Sa table de routage comporte une route à laquelle est associée une distance 1. Cette route fait partie des informations de routage diffusées périodiquement. La passerelle G2 apprend la route de Gl, l'inscrit dans sa table de routage et propage les informations relatives à cette route, en indiquant une distance 2. Enfin, G3 apprend la route de G2 et propage les informations relatives à cette route en indiquant une distance 3.

Supposez maintenant que l'accès de G1 au réseau 1 tombe en panne. G 1 met immédiatement à jour sa table de routage et affecte une distance infinie (16) à 1a route correspondante. Dans la diffusion d'informations suivante, G1 diffuse le coût élevé associé à cette route. A Moins toutefois que le protocole ne comporte des mécanismes supplémentaires pour éviter l'apparition de ce phénomène, une autre passerelle peut, avant G1 propager des informations relatives à cette route. Supposez., en particulier, que G2 diffuse ses informations de routage juste après que l'accès de G1 au réseau 1 est tombé en panne. Dans ce cas, G 1 reçoit des messages de G2 et applique l'algorithme de routage à vecteurs de distance: elle constate que G2 l'informe de l'existence d'une route vers le réseau 1 dont le coût est inférieur au sien et en déduit que le coût d'accès au réseau 1 est de 3 sauts (2 sauts pour atteindre le réseau I depuis G2 plus un pour l'atteindre depuis Gl). Gl inscrit donc dans sa table de routage la route qui passe par G 2 pour atteindre le réseau 1.
Les diffusions RIP ultérieures de ces deux passerelles ne résolvent pas rapidement le problème. Au cycle de diffusion d'informations de route suivante, G1 diffuse le contenu de l'entrée de sa table. Lorsque G2 apprend que la route vers le réseau 1 a un coût de 3, elle recalcule le coût associé à cette route, celui-ci prend la valeur 4. au troisième tour, G1 reçoit de G2 des informations qui signalent une augmentation du coût associé à la route vers le réseau 1. Elle augmente donc la valeur dans sa table de routage. Celle-ci vaut maintenant 5. et elles poursuivent ce processus jusqu'à atteindre la valeur infinie de RIP.

Protocoles utilisés pour les grands réseaux

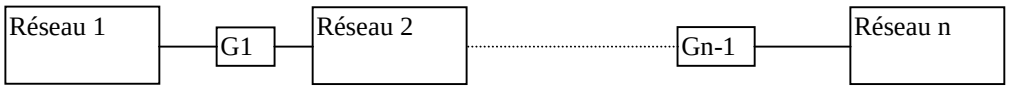
RIP est simple à mettre en place mais ne résout pas tous les problèmes.
Deplus vue le nombre de réseaux, les tables de routage RIP peuvent devenir énormes. En plus des informations statiques, généralement on utilise une route par défaut.
Lorsque l'on envoie un datagramme à une passerelle, cette dernière regarde sa table de routage pour voir si elle connaît la prochaine passerelle pour atteindre le réseau. Si oui elle lui remet de datagramme sinon, elle remet ce dernier à une passerelle spécifique qui procédera de même. Généralement, les passerelles RIP connaissent toutes les routes du réseau de l'entreprise et ignorent les routes vers les réseaux extérieures.

Elles transmettent donc tous les datagrammes destinés à l'extérieur à une autre passerelle. Cette passerelle est souvent appelée passerelle extérieurs par opposition à "passerelles intérieures". Les passerelles extérieures vont utiliser d'autres algorithmes tels que SPF(Link-state, Shortest First).
Ce protocole suppose que toutes les passerelles l'utilisant connaissent la topologie de l'ensemble des réseaux qu'elles gèrent.
EGP (Exterior Gateway Protocol) est un protocole qui est souvent mis en œuvre sur des passerelles qui font l'interconnexion de sites avec des réseaux fédérateurs. Le principe d'EGP est simple. Chaque passerelle ne connaît que ces voisins immédiats et met en place une route par défaut sur l'un de ces voisins pour pouvoir router les paquets vers des réseaux qu'elle ne connaît pas.

Toutes ces méthodes permettent donc à IP de trouver un chemin lorsqu'il existe. Cependant il existe aujourd'hui encore des cas qui posent problèmes.

La fragmentation.

Le but d'IP est de trouver un chemin pour envoyer un datagramme. Ce datagramme va circuler de passerelles en passerelles. Ces passerelles sont connectées sur un support physique qui peut avoir des MTU (Maximum Transfert Unit) différent (c'est-à-dire qui échange des trames de longueurs différentes).



Le réseau1 dispose d'un MTU M1, il est connecté au réseau 2, via G1, qui dispose d'un MTU M2, qui ... via Gn-1, qui dispose d'un MTU Mn.

Supposons qu'une machine du réseau 1 envoie un datagramme IP de longueur L à destination d'une machine sur le réseau N, alors 5 cas de figures peuvent se présenter:
1°) L < min(M1,M2,...Mn)
alors, le datagramme est émis de passerelles en passerelles jusqu'à ce qu'il atteigne sa destination sur le réseau N.
2°) L > min(M1,M2,...Mn)
alors si le datagramme ne doit pas être fragmenté, un message ICMP d'erreur est émis vers la machine source et le datagramme est détruit par la passerelle qui ne peut pas le faire transiter sur l'autre réseau.
3°) L > min(M1,M2,Mn)
alors si le datagramme peut être fragmenté, la passerelle qui ne peut émettre directement ce datagramme va le couper en autant de petits datagrammes que nécessaire et émettre tous les fragments sur l'autre réseau. Lorsque les fragments arrivent sur la passerelle suivante, cette dernière ignore que ce sont des fragments, et les traite comme des datagrammes normaux.

4°) le datagramme arrive sur une passerelle qui ne peut le traiter faute de ressources suffisantes, alors ce dernier est détruit sans autre forme de procès.
5°) le datagramme arrive sur la passerelle Gi qui ne dispose pas d'information pour router ce datagramme, alors elle le détruit et émet un message ICMP qui signale une erreur de routage.

Conclusion

IP envoie des datagrammes de machines à machines.
IP garantie qu'il fera tout son possible pour envoyer le datagramme (Best effort).
IP peut détruire un datagramme.
IP ne garantie pas qu'un datagramme émis arrive à l'identique sur l'autre machine. Il peut fragmenter le datagramme et émettre ces fragments sur différents chemins en fonction des tables de routages.
IP n'est pas un protocole fixe, mais est en perpétuel évolution.
IP ne fixe pas seul les routes, il utilise d'autres protocoles (GGP,RIP, …)

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

Cours 5 : Couche transport et résolution de noms

But

Jusqu'à présent, nous avons vu comment on pouvait envoyer des informations de machines à machines en trouvant les routes .
Lorsqu'une information arrive sur une machine, l'information est destinée à un applicatif qui fonctionne sur cette machine.
Le problème est donc d'identifier cet applicatif.

Identification des applicatifs

Un même applicatif peut être vu de façons différentes en fonction du système d'exploitation sur lequel il fonctionne. On pourra parler de programme, de processus, de job, d'application,... Chacun des noms et des identifiants associés à l'apllicatif étant différent en fonction de l'OS.
Nommer un applicatif sur une machine distante est donc difficile. De plus, l'identifiant de l'applicatif en fonction des systèmes peut varier avec le temps.
Un applicatif sur une machine peut également rendre des services différents, le problème est dans ce cas de nommer la partie de l'applicatif que l'on veut joindre. Pour ces raisons, nommer directement des applicatifs que l'on veut joindre sur une machine n'est pas une bonne idée. On utilise plutôt des numéros avec une technique de *rendez-vous*.

Les ports TCP/IP

Un port TCP/IP est un numéro de *service*. Quel que soit l'OS sur une machine, gérer des numéros, n'est pas difficile à faire. Un port doit être vu comme un lieu de rendez-vous. Le programme *serveur*, va demander au système d'exploitation de lui donner toutes les informations qui arrivent sur un ou plusieurs ports donnés. Le programme *client* qui veut dialoguer avec le serveur, doit donc émettre ses données vers un port spécifié sur une machine donnée.
Un client n'a donc aucune idée de l'appellation de l'applicatif (programme, job, processus,...).
Le problème pour un client est donc de connaître le numéro de port qui lui permettra de joindre le bon serveur. Sur la machine destinataire, un utilisateur mal intentionné (s'il dispose de privilège suffisant au niveau de l'OS) peut donc détourner des informations qui sont normalement destinées à un serveur particulier.

Affectation des ports

La question pour un client qui veut joindre un serveur spécifique sur une machine est donc de connaître le numéro de port qu'utilise ce serveur sur cette machine. La norme TCP/IP a divisé les ports en 2 catégories :
les ports libres et les ports affectés/réservés (*Well Know Ports*).
Les numéros de ports qui sont entre 0 et 2048 sont affectés et/ou réservés. Ainsi, le client qui veut joindre un serveur pour faire du transfert de fichier doit envoyer sa demande sur le port UDP 69 pour joindre un serveur utilisant le protocole *TFTP*. Les ports dont le numéro est compris entre 2048 et 65536 sont dits libres. Libre ne signifie pas qu'ils ne sont pas utilisés, mais qu'on ne peut pas connaître les serveurs (s'ils existent) qui vont répondre sur ces ports.

Etablissement d'une communication client serveur

Nous venons de voir comment on peut identifier un serveur sur une machine. Il faut maintenant voir comment un client va échanger ses données avec le serveur.
Lorsque qu'un client veut communiquer avec un serveur, il demande à son OS de lui donner un numéro de port. Ainsi, les données partent d'un port d'une machine source vers un port sur une machine destination.
Le numéro de port attribuer au client par le système n'est pas défini. Le système est libre d'attribuer le port qu'il veut.

Protocoles normalisés de la couche transport

UDP

UDP (User Datagramm Protocol) (et non pas Unix Dispense de Penser:-) UDP est un protocole de transport qui est très proche d'IP. UDP permet d'échanger des informations (USER DATAGRAMM) entre des applications. UDP prend le datagramme de l'utilisateur et le transmet à la couche IP. Cette dernière l'achemine sur la machine destinataire pour le remettre au protocole UDP. Ce dernier redonne le datagramme au processus distant. Comme UDP se contente de donner le datagramme à IP et ne fait **aucun** contrôle, il n'est pas sur que le datagramme arrive à destination, et que s'il y arrive, il n'est pas sur qu'il soit intact. Il peut avoir été fragmenté par les passerelles, les fragments ne seront pas réassemblés par UDP sur la machine du destinataire. Il est possibles que des fragments n'arrivent jamais, ou qu'ils arrivent dans le désordre. C'est aux applicatifs utilisant UDP de faire tous ces contrôles.
UDP est très peu sécurisé, il a été écrit et normaliser car il est très simple à mettre en œuvre. Sa simplicité permet de l'utiliser pour télécharger des OS sur des machines. UDP ouvre et referme une connexion pour chaque datagramme.

TCP

TCP (Transport Control Protocol) est un autre protocole de la couche transport de la famille TCP/IP. On dit que TCP est un protocole de bout en bout (END to END). Lorsque deux applicatifs utilisent TCP pour échanger des données, l'émetteur est sur que le récepteur reçoit exactement les données qui sont émises. TCP gère les contrôles. Ce sont les logiciels TCP qui redemande la transmission de paquets lorsque ces derniers ne sont pas arrivés sur le destinataire. Il assure également la remise dans l'ordre des paquets échangés. Si TCP s'appuie sur IP, il tente d'en corriger les défauts.
Le fonctionnement interne de TCP n'est pas trivial. Il procède par acquittement des messages envoyés. Pour optimiser le transfert TCP utilise une fenêtre glissante sur le bloc de données qu'il doit envoyer. La taille de la fenêtre fait l'objet de négociations entre l'émetteur et le récepteur. TCP est dit protocole en mode connecté, car lorsque qu'un canal est ouvert entre un client et un serveur, ce dernier reste valide jusqu'à sa fermeture (qui doit être demandé par au moins l'un des deux applicatifs). Le numéro de port affecté au client par son OS est donc réservé durant toute la connexion TCP, que l'applicatif envoie ou non des informations.
Tout comme UDP, pour identifier un service sur la machine distante TCP utilise des ports.

La résolution de noms

Maintenant, nous savons comment un applicatif client et un applicatif serveur échange des informations.
Ils utilisent via les protocoles TCP ou UDP des numéros de port pour différencier les différents services et des adresses IP pour identifier la machine. Si ces informations numériques sont très pratiques à traiter par des machines elles le sont moins par des hommes.
TCP/IP ne donne pas de méthode pour associer un nom à un numéro de port. Cependant beaucoup de systèmes permettent de le faire (fonction getservbyname par exemple). La correspondance entre un nom de machine et une adresse IP est maintenant bien normalisée, elle a d'ailleurs donné lieu à des protocoles qui utilisent le *DNS*. Lorsque l'on parle de résolution de nom on entend généralement par-là, la relation entre un nom de machine (*hostname)* et son adresse IP. Pour que ce système fonctionne, il faut impérativement qu'il y ait au plus une adresse IP qui corresponde au nom d'une machine.

Historique de la gestion des *hostnames*

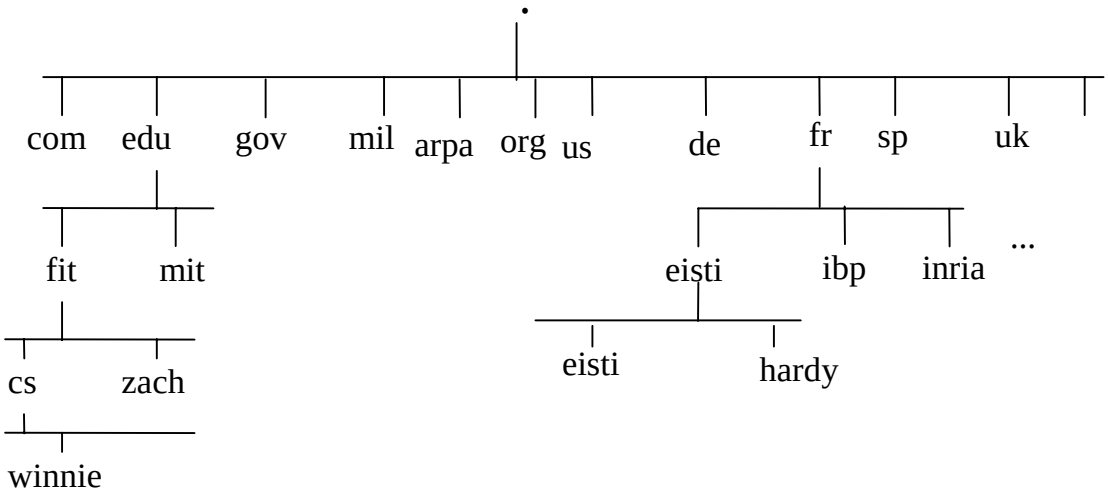
A l'origine de TCP/IP très peu de machines étaient connectées. Les administrateurs de ces machines géraient des tables de conversion manuelle. Ces tables étaient souvent des fichiers ASCII que l'on utilise encore (voir fichier /etc/hosts sous Unix). Le format de ce fichier est relativement simple:
"une adresse " "un nom de machine" "alias" "autre alias" ....
exemple :

```
194.57.186.2        enterprise     u6065
194.57.186.3        eisti    voyager        www   mail
...
```

Cette méthode est très convenable pour un réseau sur lequel il y a très peu de machines. Chaque machine doit disposer en local de cette base. Cependant, lorsque qu'un administrateur décide de changer l'adresse d'une de ces machines, il doit mettre à jour les tables sur toutes les machines de sont réseau. Plus le nombre de machines est grand sur le réseau et plus ce système devient lourd à gérer. Sur des réseaux constitués de plusieurs dizaines de machines, ce système n'est plus envisageable, car beaucoup trop lourd à maintenir. Sun, a intégrer dans sons système NIS (Network Informations Services) une base de données unique (éventuellement dupliquée sur d'autre machine). Chaque client qui veut connaître l'adresse d'une machine doit donc la demander au serveur NIS. Cette méthode présente l'avantage d'une grande souplesse. L'administrateur ne doit plus mettre à jour qu'une seule base de données et configurer l'ensemble de ces machines pour qu'elles interrogent le serveur pour obtenir l'information. Sur un réseau constitué de plusieurs centaines de machines voir de plusieurs millions (cas de l'Internet) ce système ne convient plus, car il est très difficile d'obtenir des noms différents pour chacune des machines. Ce système suppose qu'il y ait au moins une personne (physique ou morale) qui enregistre l'unicité du nom. Sur un réseau tel que l'Internet, vu le nombre de machines qui sont connectées (donc qui demande un nom et une adresse), le nombre de machines qui sont déconnectées, ou le nombre de machines dont l'adresse et/ou le nom change, un système avec une autorité centrale ne peut plus être envisagé. Il a fallut inventer un processus qui délègue une partie de la responsabilité, tout comme il a été fait pour la gestion des adresses IP. La mise en place de ce système est connue sous le nom de *DNS* (Domaine Name System) ou de Système de Nom de domaine.

Le DNS

Le système de nom de domaine est un système de noms hiérarchisés par opposition à un système de noms à plats.



Ce schéma représente une partie de l'arborescence du DNS.

Le nom qualifié ou complet (*FQDN*) d'une machine se lit en partant de la feuille et en remontant dans l'arbre. Chaque niveau est séparé par un "." Ainsi la machine sur laquelle vous avez programmé s'appelle *hardy.eisti.fr*. Le domaine racine n'a pas de nom et par convention est appelé ".". Chaque niveau de l'arborescence garantie que les noms de ces fils sont uniques. La machine qui s'appelle hardy.eisti.fr. est donc différente d'une machine qui pourrait s'appeler hardy.fit.edu. Un nom de domaine est constitué par une suite de noms séparés par des points. Le système DNS ne fait pas de différence dans sa notation entre une machine (hardy.eisti.fr.) et un domaine (eisti.fr.). Pour chaque domaine et sous domaine (., com., edu, ..., fr, ..., fit.edu, eisti.fr, ibp.fr,..) un responsable est désigné. C'est à ce responsable d'assurer l'unicité des noms du domaine qu'il gère. Ce système est distribué, car à aucun endroit de l'Internet il existe une base de données complète du système de nom.
Lorsqu'un applicatif veut connaître l'adresse IP d'une machine, il doit demander au *résolveur* (appelé par la fonction C gethostbyname) de lui la donner.
Le fonctionnement du résolveur est très simple. Il est basé sur le modèle client serveur. Dans chaque domaine, il y a une (ou plusieurs) machine qui connaît l'adresse de toutes les machines de son domaine et les adresses des serveurs de ses sous-domaines. Ainsi lorsque le résolveur doit trouver le nom d'une machine (A.B.C.D), il commence par demander au serveur de la zone (ou domaine) . l'adresse de la machine A.B.C.D.. Si le serveur de la zone "." ne connaît pas directement l'adresse, il pourra donner au résolveur l'adresse du serveur de la zone D. .

Ce dernier pourra répondre ou donner l'adresse du serveur de la zone C.D. ainsi de suite jusqu'à ce que le résolveur est obtenu l'adresse de la machine A.B.C.D.

Le système DNS est donc un système qui garantie l'unicité des noms sur le réseau Internet. Ce système ne fait pas que la résolution des noms, il peut contenir d'autres informations. Il peut par exemple donner le nom de la machine qui a pour adresse 194.57.186.17 ou donner le nom de la machine qui gère le mail d'un domaine,...
Le niveau principal de la hiérarchie du DNS n'a pas été choisi au hasard. Le découpage est normalement géographique. Chaque pays est représenté par son code de pays normalisé ISO (fr, uk, sp, us). A ceci s'ajoute quelques exceptions. Le domaine arpa où l'on a regroupé les machines qui disposaient de noms avant la mise en place du DNS.
Le domaine us existe mais aux USA, il est possible de faire enregistrer un domaine en fonction du secteur d'activité de l'établissement.

- Com pour les entreprises commerciales (dénomination internationale maintenant)
- Edu pour l'éducation
- Gov pour les institutions gouvernementales
- Mil pour les militaires
- Org pour les institutions non gouvernementales

Un sous-domaine (ou machine) peut se faire enregistrer dans plusieurs domaines.

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

Cours 6 : Réseaux industriels (généralités)

But du cours

Nous allons aborder dans ce cours des problèmes de réseaux industriels.

Différence entre un *réseau* et un *réseau industriel*

Un réseau industriel joue le même rôle qu'un réseau normal. Le but premier est toujours de transmettre des informations entre plusieurs machines. Lorsque l'on parle de réseaux, on sous-entend généralement que les machines sont des ordinateurs. Lorsque l'on parle de réseaux industriels, il s'agit de faire communiquer des machines qui ne sont plus seulement des ordinateurs. On fait communiquer des appareils différents tels que des ordinateurs, des automates programmables, des appareils de mesures, des équipements spécifiques (fours, commandes numériques, ascenseurs, ...).
Le qualificatif d'industriel pour un réseau sous-entend également un environnement particulier. L'environnement d'un réseau industriel est en général un environnement perturbé. C'est environnement est souvent pollué par des ondes électromagnétiques provenant des différents appareils (moteurs, courants forts, champs magnétiques,...). Tous ces phénomènes sont à prendre en compte dans la couche 1 du modèle OSI.

Liaison série ou liaison parallèle.

Pour relier 2 machines entre elles il existe plusieurs méthodes. L'une est dite *liaison parallèle*. Elle consiste à envoyer plusieurs informations élémentaires en même temps en utilisant plusieurs conducteurs. Cette méthode est très peu utilisée. L'autre est dite *Liaison Série*. Les données sont envoyées bit par bit les uns à la suite des autres (ex: liaison série des PC, Ethernet,...)

Liaison série asynchrone

Les liaisons séries asynchrones sont très utilisées. La plus connue est celle qui est utilisée sur les PC. Asynchrone signifie que les données sont envoyées de l'émetteur vers le récepteur sans négociation préalable. C'est au récepteur de se synchroniser sur l'émetteur. Pour ce faire l'émetteur doit envoyer un bit de START ses données(de 5 à 8 bits) suivies ou non d'un bit de parité et de 1 ou plusieurs bits de stop. Pour qu'une liaison série fonctionne, il est nécessaire de configurer les 2 extrémités pour que qu'elles utilisent la même parité, le même nombre de bits de stop (1 1,5 ou 2) la longueur des données échangées (5,6,7, ou 8 bits).

La norme RSC232 définit les valeurs des tensions que doivent fournir et reconnaître les interfaces séries des matérielles.
Un 0 logique est reconnu pour une tension allant de +8 à +40V.
Un 1 logique est reconnu pour une tension allant de -8 à -40V.
Généralement, les signaux envoyés sont compris entre -12 et + 12 V.
Sur une liaison série au repos on doit observer un 1 logique.
Pour faire un échange de données bidirectionnel entre 2 liaisons séries RS232C il faut au minimum 3 fils.
Un pour les données qui circulent dans un sens.
Un pour les données qui circulent dans l'autre sens.
Un pour la masse électrique des signaux.
Cette liaison a 3 fils est une liaison minimum. Elle nécessite une collaboration logicielle active entre les 2 machines pour contrôler le transfert des informations. Un mécanisme souvent utilisé est le protocole *XON XOFF*.

La liaison série RS232C sur PC connecteur 25 poi**nts**

Broche	NOM	Sens
1	FG	
2	TX	Sortie
3	RX	Entrée
4	RTS	Sortie
5	CTS	Entrée
6	DSR	Entrée
7	SG	
8	DCD	entrée
20	DTR	sortie
22	RI	entrée

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

La liaison série RS232C sur PC connecteur 9 p**oints**

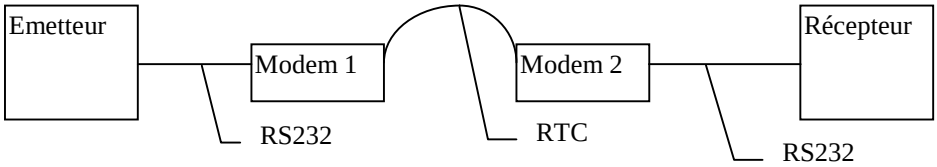
Broche	NOM	Equivalence en db 25
1	DCD	8
2	RX	3
3	TX	1
4	DSR	20
5	SG	7
6	DSR	6
7	RTS	4
8	CTS	5
9	RI	22

Description des signaux

FG : Masse châssis
TX : Transmission des données
RX : Réception des signaux
RTS : Demande de Transmission
CTS : Prêt à émettre
DSR : Emetteur prêt
SG : Masse Electrique
DCD : Détection de porteuse
DTR : Terminal prêt
RI : Indicateur de sonnerie.

Le signal FG est utilisé par des câbles blindés.
Le signal SG est la masse électrique qui doit être utilisée pour comparer les valeurs des autres signaux.
Sur la broche TX, l'émetteur émet les données
Le signal RTS est positionné par l'émetteur lorsque ce dernier veut émettre des données.
Le signal DTR est positionné par l'émetteur pour signaler au récepteur qu'il est en ligne.
Sur la broche RX, le récepteur reçoit les données.
Le signal RTS indique à l'émetteur que son correspondant est prêt à recevoir des données.
Le signal DSR indique que l'émetteur est toujours présent.
DCD indique que la porteuse est présente.
RI indique une sonnerie.

A première vue ces signaux peuvent paraître redondants ou inutiles pour une communication entre 2 machines, mais ils sont très pratiques pour une communication entre 2 machines reliées par le RTC via des modems utilisant un protocole matérielle.



Sur TX/RX, circulent les données qui sont échangées entre l'émetteur et le récepteur.
RTS et CTS fixe la communication entre l'émetteur et son modem (ou le récepteur et son modem)
DSR sur l'émetteur signifie que le récepteur est toujours présent.
DCD indique à l'émetteur que la porteuse utilisée sur le RTC est présente (donc que le modem 2 n'a pas raccroché).
DTR indique au modem que l'émetteur est près à échanger des informations avec le modem.
RI indique au récepteur que le modem détecte une sonnerie.

Différents câblages RS232

Câblage DTE /DCE

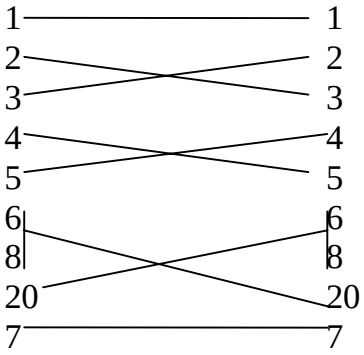
Pour relier 2 équipements via une liaison série la norme RS232 prévoit 2 brochages différents (DTE et DCE). Le brochage type DTE (Data Terminal Equipement) doit être utilisé pour des équipements terminaux. Le brochage type DCE (Data Control Equipement) est normalement utilisé pour des équipements intermédiaires utilisés sur des liaisons (modems,...). Le brochage DTE normalisé est, celui décrit sur les connecteurs à 25 points des PC. Le brochage DCE est très simple, sur le connecteur DB25 de l'équipement on retrouve les mêmes signaux en les croisant.
Pour relier un équipement DTE (PC, terminal, imprimante,..) à un équipement DCE (modem), il suffit donc de relier la broche 1 à la broche 1,.... la broche 22 à la broche 22.
Comment différencier un équipement DTE d'un équipement DCE sans documentation?
Astuce : au repos (sans connexion) un 1 logique doit être observé sur la broche TX. Si l'équipement est DTE on doit donc trouver une tension négative entre la broche 2 et la broche 7 du connecteur 25 points. Si l'équipement est de type DCE, la tension négative doit se trouver entre les broches 3 et 7.

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

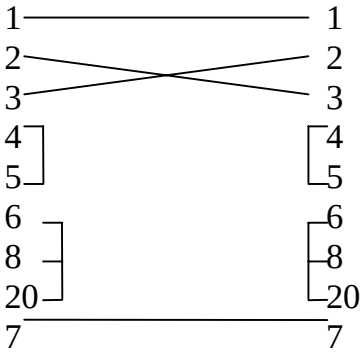
Câblage DTE/DTE

IL n'est pas toujours nécessaire d'utiliser des modems pour relier deux équipements par des RS232C. La norme RS232C spécifie que l'on doit pouvoir(avec des câbles blindés et de tension de 40V) atteindre 150m . En pratique avec une distance supérieure à 10 ou 15m on commence à avoir des problèmes. Si les équipements sont à moins de 10 m, il est tout à fait possible de les relier sans passer par modem. Il s'agit de réaliser un câble *Null Modem*. Malheureusement il existe une multitude de types de câbles Null modem. Il faut savoir si les équipements terminaux utilisent un protocole matériel (type RTS/CTS) ou un protocole logiciel (type XON XOFF)

Câble PC - PC en DB25 utilisant RTS/CTS



Câble PC - PC en DB25 utilisant XON/XOFF



Différents protocoles

Protocole Matériel

Un des protocoles le plus utilisé est celui qui tous les signaux de la RS232 Lorsque l'émetteur veut émettre ses donnés, il doit positionner la ligne RTS pour demander aux récepteur s'il est prêt à accepter ces données. Le récepteur lorsqu'il est prêt à recevoir les données va envoyer le signal DSR de l'émetteur pour lui indiquer qu'il est prêt. Lorsque l'émetteur veut suspendre l'émission, il va enlever le signal DSR de l'émetteur.

Protocole XON/XOFF

Ce protocole ne nécessite qu'une liaison sur 3 fils. Le reste de la négociation entre l'émetteur et le récepteur pour échanger des données ce fait par logiciel. Ce protocole est basé sur les caractères XON (ASCI 11H) et XOFF (ASCI 13H).

Le récepteur gère un buffer. Lorsque son buffer est plein à 80 % le récepteur envoie le caractère XOFF. L'émetteur lorsqu'il reçoit le caractère XOFF doit immédiatement suspendre son émission. Lorsque l'émetteur a vidé sont buffer à 50% il envoie un caractère XON à l'émetteur. A la réception de XON l'émetteur peut reprendre son émission. Il est possible que l'émetteur ne reçoive pas ou perde les caractères XON/ XOFF. Pour pallier à ces problèmes, lorsque l'émetteur n'a pas reçu de caractères depuis un certain temps, ce dernier peut reprendre de sa propre initiative le transfert. Si le récepteur n'est pas d'accord, ce dernier pourra toujours réemmettre un XOFF.

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

Cours 7 : Vocabulaire & différents problèmes

But du cours

Nous avons jusqu'à présent basé l'étude des réseaux sur des problèmes techniques particuliers. Aujourd'hui, nous allons voir des concepts généraux sur les réseaux et voir quelles en sont les avantages et les inconvénients.

Serveur dédié ou non dédié

Lorsque l'on parle de serveurs sans autre précision sur un réseau, on parle de serveurs de fichiers. Un réseau à serveur dédié est un réseau sur lequel au moins une machine n'est pas utilisée comme station de travail. C'est le cas des réseaux Netware de Novell. Sur ce type de réseau au moins une machine (le serveur) fonctionne sur un système particulier (Netware) tandis que les autres continuent à travailler sur le système d'exploitation "normal"(PC sous dos, Windows, OS2). Sur le serveur, on ne peut faire fonctionner que des programmes spécifiques. Sur un réseau sans serveur dédié toutes les machines sont susceptibles de permettre le partage de leurs fichiers aux autres stations (réseau Windows for WorkGroup, Windows NT,...).

Poste maître / esclave

Sur certains réseaux (souvent des réseaux Automates Programmables) on trouve une notion de poste maître et de poste esclave. Le poste maître est un des postes sur le réseau qui va échanger des informations avec les esclaves. En aucun cas, une communication d'esclave à esclave ne sera possible.

Collecte d'informations en *polling* ou en *selecting*

Un fonctionnement en polling est établi lorsqu'un processus (ou machine) va interroger les autres pour savoir si ces derniers ont des informations à échanger avec celui-ci (ce mode est celui qui est utilisé dans des réseaux de type maître / esclave). Un fonctionnement en selecting est différent. Le processus qui collecte les demande d'informations attend que d'autres processus (ou machine) lui demande quelque chose (fonctionnement de type client / serveur). On parle de selecting, car le serveur doit "choisir " d'où vient la demande et voir comment y réagir.

Problèmes de partage des données

Dans une architecture client serveur, on peut avoir des problèmes lorsque des clients différents peuvent vouloir manipuler les mêmes données d'un serveur. Il est possible d'aboutir à des incohérences (donner la même place à deux clients dans un train, problème pour accéder à des fichiers,...).
Lorsque plusieurs machines partagent des espaces disques ensemble via un réseau (avec nfs, vfs, ...) il faut savoir quelle stratégie utiliser lorsqu'un incident réseau arrive.

Problèmes de relations humaines

Dans une entreprise, il arrive que plusieurs systèmes (ou réseaux) autonomes soient reliés un jour ensemble. Dans ce cas, il se pose des problèmes. Dans chaque partie autonome un ou plusieurs administrateurs étaient "libres". Ils pouvaient prendre toutes les décisions qu'ils voulaient concernant la gestion de leurs systèmes. Lorsque ces systèmes sont interconnectés, la marge de manœuvre des administrateurs locaux se trouve réduite. L'ensemble de ces administrateurs locaux doivent collaborer, ensemble et avec l'administrateur du réseau. La prise d'une décision sur un des environnements autonomes peut avoir des conséquences sur toute l'organisation.

Problèmes techniques

Time Out

Lorsque l'on fait de la communication en réseau on doit toujours prendre en compte le fait qu'entre la réception de 2 morceaux des données échangées il y aie une rupture de la liaison. En programmation client / serveur on risque de bloquer les processus. Pour éviter ce problème on doit mettre en place un système de *watch dog*. Avant une lecture ou une écriture on demande poliment au système de bien vouloir avoir l'extrême obligeance de nous rappeler dans N secondes sauf avis contraire. Si le système redonne la main au processus qui la demandé, le programmeur peut donc prévoir une gestion des erreurs.

Cohérence des données

Dans une application des données ne peuvent être valide que si elle respecte entre elles une certaine formule. Ex: 3 variables A, B, C avec C = A+B. Si un client modifie l'un des variables A ou B il doit également modifier C. Que se passe-t-il si entre la mise à jour d'A (ou de B) et de C il y a une rupture de ligne. Notre base d'informations n'est plus cohérente puisque à cet instant C <> B+C. Lorsque l'on programme ce type d'application, il est nécessaire de prévoir des mécanismes de retour arrière et/ou de reprise sur incidents pour conserver la validité des données.

Performance du réseau

Lorsque l'on utilise un réseau, on doit prendre en compte ses caractéristiques physiques (en particulier son MTU). Imaginons une station qui gère des capteurs (200). Régulièrement (toutes les secondes) cette station doit envoyer à un serveur la valeur de ces capteurs. Si l'échelle possible des valeurs pour les capteurs est grande (EX : température d'un four de 0 à 10000 degrés). Il faut donc au minimum (en supposant que cette station soit seule sur son réseau) un MTU de 2 * 200 *8 = 3Kb/s (2, car 2 octets sont nécessaire pour stocker 10 000) uniquement pour transmettre les données sans compter les entêtes de trames résultant de l'encapsulation. Avec ce phénomène, on peut arriver à saturation du réseau. Avec un peu de réflexion, on peut s'apercevoir que la variation de température dans une seconde du four n'est que de quelques degrés. Plutôt que de transmettre la température, il est plus économique de transmettre sa variation par rapport à la mesure précédente. Ainsi, on peut très largement optimiser le transfert sur le réseau.

Représentation des données

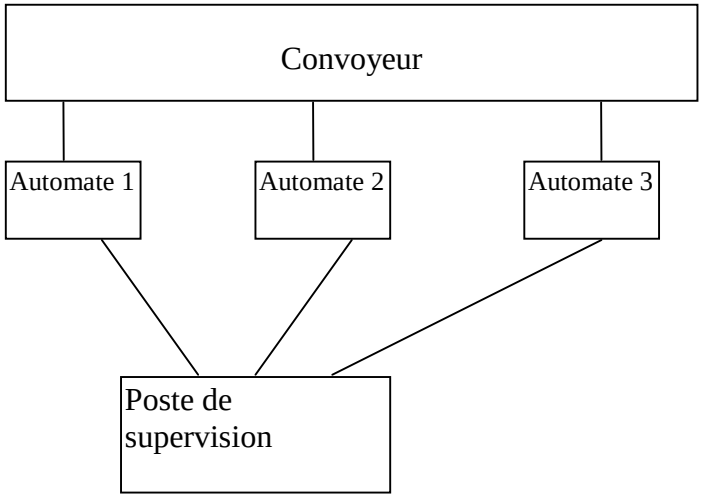
Lorsque l'on communique des informations binaires sur un réseau entre des machines, il peut se poser un problème pour la représentation de l'information. Une machine peut par exemple stocker d'abord les poids forts de l'information (Big Endian) et l'autre stocker d'abord les poids faibles(Little Endian). Si ces 2 machines décident d'échanger des informations il faut que ces machines se mettent d'accord sur la manière dont elles vont échanger les informations. Pour résoudre ce problème la technique la plus utilisée est la mise en place d'un alphabet réseau. Chaque machine doit convertir ses informations dans cet alphabet avant de les véhiculer sur le réseau. La machine destinataire devra retransformer les données de l'alphabet réseau dans son propre système.
Le même problème se retrouve pour des données de longueur variable, certain système utilise un compteur et d'autre une donné particulière pour indiquer la fin.

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

TP 1

Présentation du sujet

Il s'agit de simuler le déroulement d'un processus industriel dont voici le schéma.:



Sur le convoyeur vont circuler des pièces qui seront transformées par les différents automates.
L'automate 1 est informé de la présence d'une pièce par l'opérateur. Il prend la pièce et lui fait subir une première modification. Lorsque la modification est effectuée, il pose sa pièce sur le convoyeur, et informe l'automate 2 qu'une pièce est disponible pour lui sur le convoyeur.
L'automate 2, grâce à l'information communiquée par l'automate 1, prend la pièce, lui fait subir sa seconde transformation , la redépose sur le convoyeur et informe l'automate 3.
L'automate 3, grâce à l'information communiquée par l'automate 2, prend la pièce, lui fait subir sa troisième transformation et informe l'opérateur à la fin de l'opération.

A chaque arrivée et départ de pièce sur un automate, ce dernier enverra un message au poste de supervision. Le poste de supervision stockera ces messages sur disque pour un éventuel traitement ultérieur (statistiques, ...)

Informations techniques

Les automates pourraient être dans la réalité des automates programmables ou des machines spécifiques. Il faut cependant qu'ils soient reliés à un réseau local avec le poste de supervision.
Nous supposerons que le *protocole* réseau est *TCP/IP*.

Simulation Informatique

Chaque automate, ainsi que le poste de supervision, sera simulé par un *processus* sur une machine UNIX. Pour des questions de disponibilité des machines, tous les processus fonctionneront sur la même machine, cependant avec les techniques que nous allons mettre en œuvre, les différents processus pourraient s'exécuter sur des machines différentes (à condition que ces machines soient reliées par un réseau utilisant TCP/IP).

Glossaire

Ce glossaire se veut simple, il a simplement pour but de définir rapidement les quelques termes qui apparaissent dans ce document. Ces termes, seront revus et détaillés durant les cours.

Network (Réseau)

Ensemble de machines interconnectées de manière à échanger des informations. On distingue plusieurs types de réseaux en fonctions de leur taille, de leur importance géographique (*LAN, WAN, MAN*,...). En fonction du contexte, le terme réseau pourra désigner le câblage ou, le câblage et le(s) *protocole(s)* utilisé(s) pour l'interconnexion.

LAN:

(Local Area Network) réseau local . Un LAN est un réseau situé généralement dans la même entité géographique (entreprise, campus,...). Des LAN peuvent être interconnectés pour former des réseaux plus grands (WAN, MAN,...). On dit alors que le LAN est un sous-réseau du réseau auquel il est connecté.

WAN

(Wide Area Network) réseau grande distance. Un WAN est un réseau qui se mesure sur une grande échelle géographique. Certaines sociétés, généralement internationales (IBM, UNISYS, AT&T, AIR France, ...) disposent souvent de tels réseaux à l'échelle planétaire.
Internet est un réseau de type WAN.

MAN
: (Metropolitan Area Network) Ce type de réseaux est récent et garde les avantages des LAN sur de plus longues distances de l'ordre de la ville.

OSI
: (Open Systems Interconnection) modèle de référence pour l'interconnexion des systèmes ouverts. Le modèle OSI est un modèle à sept couches, proposé par *l'ISO*, qui permet de représenter toutes les choses qu'un réseau doit gérer.

ISO
: (International Standardization Organisation) Organisation internationale de normalisation

TCP/IP
: (Transfert Control Protocol /Internet Protocol)TCP/IP est un nom de famille de *protocoles.* TCP et IP sont deux protocoles de cette famille, mais ne sont pas les seuls (UDP, ICMP, IGP,...). Lorsque l'on parle de "TCP/IP" on parle de la famille des protocoles TCP/IP en général, et pas des protocoles TCP et IP en particulier.

Matériels actifs
: Ensemble d'appareils dont le but est d'intervenir sur un réseau afin de le connecté à un autre ou d'en améliorer sa qualité.

Adressage
: Technique qui permet d'identifier une machine particulière sur un réseau.

Routage
: Méthode et *protocoles* qui permettent de trouver un chemin pour échanger des informations entre machines sur un réseau.

Client/Serveur
: Méthode de programmation qui permet de structurer des applications qui échangent des informations entre plusieurs machines et/ou *processus*.

Protocole
: Ensemble de règles qui régissent un échange d'informations. Un réseau pour son bon fonctionnement, suppose que les machines qui vont échanger des informations ont des règles communes sur la manière de dialoguer. Un réseau est généralement régi par plusieurs protocoles (protocoles de hauts niveaux, protocoles de bas niveaux ,...).

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

Tp 2

Présentation

Dans ce TP nous allons nous familiariser avec le matériel étudier les différentes primitives que nous allons utilisées dans la simulation.

Environnement de travail

L'environnement est constitué de *Terminaux X* sur le réseau local (Ethernet 10B2 voir cours). Sur ce réseau local on trouve une machine SUN: Sparc 20 fonctionnant sous *Unix*.
Un terminal X est un terminal *intelligent* qui communique avec la machine en utilisant le protocole X Window. Ce terminal est *intelligent* car il dispose de mémoire et d'un processeur. L'utilisation du processeur du terminal soulage celui (ou ceux) de la machine.
La machine Unix est souvent appelée *Serveur*.
Le terminal dans la terminologie X Window est appelé serveur d'affichage (il attend des ordres venant de la machine). Les clients de ce serveur sont les applications que l'on exécute sur la machine. La machine joue donc également le rôle de serveur d'application ou de fichiers.

But du TP

Etudier la syntaxe des différentes primitives nécessaires à l'écriture d'un client et d'un serveur TCP/IP.

Primitives utilisées

gethostbyname, getservbyname, socket, bind, listen, accept, connect

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

Ossature des programmes

Serveur

Voici la marche à suivre pour un serveur
Demander un socket
Demander au système de rediriger les données qui pourraient arrivées sur le port vers notre programmes.
Attendre une connexion.
Accepter (ou refuser) la connexion.
Répondre

Client

Voici la marche à suivre pour un client
Demander un socket
Se connecter au serveur
Communiquer.

Travail demander

A l'aide du manuel, repérer les "include" nécessaire au fonctionnement des différentes primitives.
Etudier les paramètres des différentes primitives.
Réaliser l'écriture d'un serveur qui affiche une chaîne de caractères émise par le client.
Réaliser l'écriture d'un client qui envoie une chaîne de caractères au serveur.

Corriger

```
/*********************************************/
/* BP le 22/02/1997                          */
/* Programme qui recoit un message sur un socket  */
/* Ossature d'un serveur ...                 */
/*********************************************/
/* Fichiers a inclure .... */
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/* Constantes */
        /* Message envoye au serveur */
#define TAILLEMAX 100 /* Pour chaine de caracteres */


/* Prototypages */
int main(int argc, char ** argv);

int main(int argc, char ** argv)
{
int Fd; /* Descripteur de fichier (socket) */
int FdAccept; /* Descripteur de fichier (socket) */
struct sockaddr_in Sockaddr; /* Descriptif d'un socket */
struct hostent * Machine; /* Machine ou reside le serveur */
struct servent * Service; /* Service (ou port) ou le serveur est attache */
char Message[TAILLEMAX +1];
int Lenght;

/* Verifier l'appel */
if (argc != 2)
        {
        fprintf(stderr,"Syntaxe : %s Port...\n",argv[0]);
        exit (-10);
        }

/* le serveur est sur la machine ou on lance le programme */
Machine = gethostbyname("localhost");
if (Machine == NULL)
        {
        perror("Serveur : gethostbyname");
        exit(-1);
        }
/* initialisation de la structure */
memset(&Sockaddr,0,sizeof(struct sockaddr_in));
/* on travaille sur TCP/IP */
Sockaddr.sin_family = PF_INET;
memcpy(&(Sockaddr.sin_addr),Machine->h_addr,Machine->h_length);

/* on ecoute sur quel port ???? */
Service = getservbyname(argv[1],"tcp");
if (Service == NULL)
```

```
		{
		perror("Serveur : getservbyname");
		exit(-2);
		}
Sockaddr.sin_port = Service->s_port;

/* On peut avoir le socket */
/* PF_INET => socket tcp/ip */
/* SOCK_STREAM => tcp comme protocol */
/* 0 => protocole par defaut sur tcp */
Fd = socket(PF_INET,SOCK_STREAM,0);
if (Fd == -1)
		{
		perror("Serveur : socket");
		exit(-3);
		}

/* reserver le port sur lequel on veut ecouter */
if (bind(Fd,(struct sockaddr *) & Sockaddr,sizeof(struct sockaddr_in)) == -1)
		{
		perror("Serveur : bind");
		exit (-4);
		}

/* on attend au maximum une connection a la fois */
if (listen(Fd,1) == -1)
		{
		perror("Serveur : listen");
		exit (-5);
		}

/* on accepte la connexion */
Lenght = sizeof(struct sockaddr_in);
if ((FdAccept = accept(Fd,(struct sockaddr *) &Sockaddr, &Lenght)) == -1)
		{
		perror("Serveur : accept");
		exit (-6);
		}

/* on tente la lecture du message */
read(FdAccept,Message,TAILLEMAX);
printf("On a recu : %s \n",Message);
close(FdAccept);
close(Fd);
exit(0);
}
```

```
/*********************************************/
/* BP le 22/02/1997                          */
/* Programme qui emet un message sur un socket */
/* Ossature d'un client tcp                  */
/*********************************************/
/* Fichiers a inclure .... */
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/* Constantes */
        /* Message envoye au serveur */
#define MESSAGE " Message du client .... "

/* Prototypages */
int main(int argc, char ** argv);

int main(int argc, char ** argv)
{
int Fd; /* Descripteur de fichier (socket) */
struct sockaddr_in Sockaddr; /* Descriptif d'un socket */
struct hostent * Machine; /* Machine ou reside le serveur */
struct servent * Service; /* Service (ou port) ou le serveur est attache */

/* verifier le lancement */
if ( argc != 3)
        {
        fprintf(stderr," Syntaxe : %s : Machine Port...\n",argv[0]);
        exit(-10);
        }
/* addresse de la machine du serveur */
Machine = gethostbyname(argv[1]);
if (Machine == NULL)
        {
        perror("Client : gethostbyname");
        exit(-1);
        }
/* initialisation de la structure */
memset(&Sockaddr,0,sizeof(struct sockaddr_in));
/* on travaille sur TCP/IP */
Sockaddr.sin_family = PF_INET;
memcpy(&Sockaddr.sin_addr,Machine->h_addr,Machine->h_length);
```

```
/* on ecoute sur quel port ???? */
Service = getservbyname(argv[2],"tcp");
if (Service == NULL)
        {
        perror("Client : getservbyname");
        exit(-2);
        }
Sockaddr.sin_port = Service->s_port;
/* On peut avoir le socket */
/* PF_INET => socket tcp/ip */
/* SOCK_STREAM => tcp comme protocol */
/* 0 => protocole par defaut sur tcp */
Fd = socket(PF_INET,SOCK_STREAM,0);
if (Fd == -1)
        {
        perror("Client : socket");
        exit(-3);
        }

/* on se connecte au serveur */
if ( connect(Fd,(struct sockaddr *) &Sockaddr,sizeof(struct sockaddr_in)) == -1)
        {
        perror("Client : connect");
        exit(-4);
        }

/* on tente la l'envoie d'un message */
write(Fd,MESSAGE,strlen(MESSAGE) +1 );
close(Fd);
exit(0);
}
```

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

TP 3

But du TP

Mettre en place l'automate 1 et 2 et 3
Nous commencerons par écrire les automates 1 et 3

Description de l'automate 1

Tant que vrai faire
début
Attendre Opérateur
Prendre Pièce
Traiter Pièce
Attendre automate 2
Répondre automate 2
fin

Description de l'automate 2

Tant que vrai faire
début
Demander une pièce à automate 1
Prendre Pièce
Traiter Pièce
Attendre automate 3
Répondre automate 3
fin

Description de l'automate 3

Tant que vrai faire
début
Demander une pièce à automate 2
Prendre Pièce
Traiter Pièce
Avertir Opérateur
Attendre opérateur
fin

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

Annexes1 Vocabulaire simplifié

Vocabulaire simplifié pour l'interface des sockets

Pour la description de programme en fonctionnement client/serveur avec TCP/IP basé sur l'interface des sockets on utilisera la symbolique suivante afin de ne pas compliquer le raisonnement avec des problèmes de syntaxe de C et/ou d'UNIX tout en restant proche de la philosophie.

Pour faire établir une communication entre un client qui fonctionne sur une machine d'adresse Ipc et un serveur fonctionnant sur une machine d'adresse Ips et qui attend des connexions sur un port P on utilisera la syntaxe suivante.

Pour le client

```
Début
        Var = socket()
        connect(Var,Ips,P)
        read(Var,Message) ou write(Var,Message)
        ....
        close(Var)
Fin
```

Pour le serveur (mono client)

```
Début
        Var = socket()
        bind(Var,Ips,Port)
        listen(Var)
        Var1 = Accept(Var)
        write(Var1,Message) ou read(Var1,Message)
        ...
        close (Var)
        close (Var1)
Fin
```

Pour le serveur (multi clients)

```
Début
        Var = socket()
        bind(Var,Ips,Port)
        listen(Var)
        Var1 = Accept(Var)
        Tant que vrai faire
        Début
                Lancer Répondeur(Var1)
                close(Var1)
        Fin
        close (Var)
Fin

Répondeur(Var2)
Début
        write(Var2,Message) ou read(Var2,Message)
Fin
```

Avec Var, Var1, Var2, Message des variables.

Avenue du Parc
95011 CERGY
FRANCE
http://www.eisti.fr

Annexe 2 :corrigés des TP

automate 1

```
/* Fichiers a inclure .... */
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/utsname.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/* Constantes */
/* Message envoye au serveur */
#define TAILLEMAX 100 /* Pour chaine de caracteres */

/* Prototypages */
int main(int argc, char ** argv);

int main(int argc, char ** argv)
   {
   int Fd; /* Descripteur de fichier (socket) */
   int Fd4; /* Descripteur de fichier (socket) */
   int FdAccept; /* Descripteur de fichier (socket) */
   struct sockaddr_in Sockaddr; /* Descriptif d'un socket */
   struct sockaddr_in Sockaddr4; /* Descriptif d'un socket */
   struct hostent * Machine4; /* Machine ou reside le serveur4 */
   struct servent * Service4; /* Service (ou port) ou le serveur4 est
attache */
   struct hostent * Machine; /* Machine ou reside le serveur */
   struct servent * Service; /* Service (ou port) ou le serveur est
attache */
   struct utsname Uname;  /* Informations sur la machine (non,
version,...) */
   char Message[TAILLEMAX +1];
   char Message4[TAILLEMAX +1];
   char Reponse[TAILLEMAX+1]; /* Reponse a Auto2 */
   char NomMachine[SYS_NMLN +1];
   int Piece;   /* Numero de la piece traitee */
   int Lenght;

   /* Verifier l'appel */
   if (argc != 4)
      {
      fprintf(stderr,"Syntaxe : %s Port Machine4
Port4...\n",argv[0]);
      exit (-7);
      }
```

```
/* Obtenir le nom de la machine sur le reseau */
   if ( uname(&Uname) == -1)
     {
     perror("Auto1 : : uname :");
     exit(-10);
     }
   strcpy(NomMachine,Uname.nodename);

   /* le serveur est sur la machine ou on lance le programme */
   Machine = gethostbyname(NomMachine);
   if (Machine == NULL)
      {
      perror("Auto1 :  : gethostbyname");
      exit(-1);
      }
   /* initialisation de la structure */
   memset(&Sockaddr,0,sizeof(struct sockaddr_in));
   /* on travaille sur TCP/IP */
   Sockaddr.sin_family = PF_INET;
   memcpy(&(Sockaddr.sin_addr),Machine->h_addr,Machine->h_length);

   /* on ecoute sur quel port ???? */
   Service = getservbyname(argv[1],"tcp");
   if (Service == NULL)
      {
      perror("Auto1 :  : getservbyname");
      exit(-2);
      }
   Sockaddr.sin_port = Service->s_port;



/* Initialisation vers Automat4 */
   Machine4 = gethostbyname(argv[4]);
   if (Machine4 == NULL)
      {
      perror("Auto1 :  : gethostbyname");
      exit(-1);
      }
   /* initialisation de la structure */
   memset(&Sockaddr4,0,sizeof(struct sockaddr_in));
   /* on travaille sur TCP/IP */
   Sockaddr4.sin_family = PF_INET;
   memcpy(&(Sockaddr4.sin_addr),Machine4->h_addr,Machine4->h_length);

   /* on ecoute sur quel port ???? */
   Service4 = getservbyname(argv[5],"tcp");
   if (Service4 == NULL)
      {
      perror("Auto1 :  : getservbyname");
      exit(-2);
      }
   Sockaddr4.sin_port = Service4->s_port;
```

```
   /* On peut avoir le socket */
   /* PF_INET => socket tcp/ip */
   /* SOCK_STREAM => tcp comme protocol */
   /* 0 => protocole par defaut sur tcp */
   Fd = socket(PF_INET,SOCK_STREAM,0);
   if (Fd == -1)
      {
      perror("Auto1 :  : socket");
      exit(-3);
      }

   /* reserver le port sur lequel on veut ecouter */
   if (bind(Fd,(struct sockaddr *) & Sockaddr,sizeof(struct
sockaddr_in)) == -1)
      {
      perror("Auto1 :  : bind");
      exit (-4);
      }

   /* on attend au maximum une connection a la fois */
   if (listen(Fd,1) == -1)
      {
      perror("Auto1 :  : listen");
      exit (-5);
      }

   Piece = 0;
   while (1)
      {
      printf(" Auto1 : Mettre une Piece (puis RETURN) \n");
      getchar();
      Piece +=1; /* On traite une piece en plus ... */
      printf("Auto1 : Traiter Piece numero %d\n",Piece);
      sleep(5);
      /* Informer auto4 */
     /* On peut avoir le socket */
     /* PF_INET => socket tcp/ip */
     /* SOCK_STREAM => tcp comme protocol */
     /* 0 => protocole par defaut sur tcp */
     Fd4 = socket(PF_INET,SOCK_STREAM,0);
     if (Fd4 == -1)
        {
        perror("Auto1 :  : socket");
        exit(-3);
        }
      if (connect ( Fd4,(struct sockaddr *) &Sockaddr4,sizeof(struct
sockaddr_in))== -1)
         {
         perror("Auto1 : connect");
         exit(-11);
         }
      sprintf(Message4,"Auto1 a fini la piece numero %d",Piece);
      write(Fd4,Message4,strlen(Message4)+1);
      close(Fd4);
```

```
        printf("Auto1 : Attendre Demande...\n");
        /* on accepte la connexion */
        Lenght = sizeof(struct sockaddr_in);
        if ((FdAccept = accept(Fd,(struct sockaddr *) &Sockaddr,
&Lenght)) == -1)
           {
           perror("Auto1 :  : accept");
           exit (-6);
           }

        /* on tente la lecture du message */
        read(FdAccept,Message,TAILLEMAX);
        /* On devrait testert la validiter de la demande .... */
        sprintf(Reponse,"%d :Voici la piece ",Piece);
        write(FdAccept,Reponse,strlen(Reponse)+1);
        close(FdAccept);
        }
     close(Fd);
     exit(0);
     }
```

automate 2

```
/* Fichiers a inclure .... */
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/utsname.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/* Constantes */
   /* Message envoye au serveur */
#define MESSAGE " Donner une piece ! "
#define TAILLEMAX 100 /* Pour chaine de caracteres */

/* Prototypages */
int main(int argc, char ** argv);
```

```
int main(int argc, char ** argv)
   {
   int Fd4; /* Descripteur de fichier (socket) */
   int Fd21; /* Descripteur de fichier (socket) */
   int Fd23; /* Descripteur de fichier (socket) */
   int FdAccept23; /* Descripteur de fichier (socket) */
   struct sockaddr_in Sockaddr23; /* Descriptif d'un socket */
   struct sockaddr_in Sockaddr4; /* Descriptif d'un socket */
   struct sockaddr_in Sockaddr21; /* Descriptif d'un socket */
   struct hostent * Machine4; /* Machine ou reside auto4 */
   struct hostent * Machine23; /* Machine ou reside le client */
   struct hostent * Machine21; /* Machine ou reside le serveur */
   struct servent * Service21; /* Service (ou port) ou le serveur est
attache */
   struct servent * Service4; /* Service (ou port) ou le serveur 4est
attache */
   struct servent * Service23; /* Service (ou port) ou le serveur est
attache */
   struct utsname Uname;  /* Informations sur la machine (non,
version,...) */
   int res; /* pour tester les fonctions */
   char Message4[TAILLEMAX +1];

   char Message[TAILLEMAX +1];
   char Reponse[TAILLEMAX+1]; /* Reponse a Auto3 */
   char NomMachine[SYS_NMLN +1];
   int Piece;   /* Numero de la piece traitee */
   int Lenght;

   /* Verifier l'appel */
   if (argc != 6)
      {
      fprintf(stderr,"Syntaxe : %s PortServeur MachineClinet
PortClient Machine4 port4\n",
                                          argv[0]);
      exit (-7);
      }

   /* Obtenir le nom de la machine sur le reseau */
   if ( uname(&Uname) == -1)
     {
     perror("Auto2 : : uname :");
     exit(-10);
     }
   strcpy(NomMachine,Uname.nodename);

   /* le serveur est sur la machine ou on lance le programme */
   Machine23 = gethostbyname(NomMachine);
   if (Machine23 == NULL)
      {
      perror("Auto2 :  : gethostbyname");
      exit(-1);
      }
   /* initialisation de la structure */
   memset(&Sockaddr23,0,sizeof(struct sockaddr_in));
   /* on travaille sur TCP/IP */
   Sockaddr23.sin_family = PF_INET;
```

```
   memcpy(&(Sockaddr23.sin_addr),Machine23->h_addr,Machine23-
>h_length);

   /* on ecoute sur quel port ???? */
   Service23 = getservbyname(argv[1],"tcp");
   if (Service23 == NULL)
      {
      perror("Auto2 :  : getservbyname");
      exit(-2);
      }
   Sockaddr23.sin_port = Service23->s_port;

   /* On peut avoir le socket */
   /* PF_INET => socket tcp/ip */
   /* SOCK_STREAM => tcp comme protocol */
   /* 0 => protocole par defaut sur tcp */
   Fd23 = socket(PF_INET,SOCK_STREAM,0);
   if (Fd23 == -1)
      {
      perror("Auto2 :  : socket");
      exit(-3);
      }

   /* reserver le port sur lequel on veut ecouter */
   if (bind(Fd23,(struct sockaddr *) & Sockaddr23,sizeof(struct
sockaddr_in)) == -1)
      {
      perror("Auto2 :  : bind");
      exit (-4);
      }

   /* on attend au maximum une connection a la fois */
   if (listen(Fd23,1) == -1)
      {
      perror("Auto2 :  : listen");
      exit (-5);
      }

  /* Initialisation de la partie Client */


   /* addresse de la machine du serveur */
   Machine21 = gethostbyname(argv[2]);
   if (Machine21 == NULL)
      {
      perror("Auto2 : gethostbyname");
      exit(-1);
      }
   /* initialisation de la structure */
   memset(&Sockaddr21,0,sizeof(struct sockaddr_in));
   /* on travaille sur TCP/IP */
   Sockaddr21.sin_family = PF_INET;
   memcpy(&Sockaddr21.sin_addr,Machine21->h_addr,Machine21-
>h_length);
```

```
   /* on ecoute sur quel port ???? */
   Service21 = getservbyname(argv[3],"tcp");
   if (Service21 == NULL)
      {
      perror("Auto2 : getservbyname");
      exit(-2);
      }
   Sockaddr21.sin_port = Service21->s_port;
/* Initialisation vers Automat4 */
   Machine4 = gethostbyname(argv[4]);
   if (Machine4 == NULL)
      {
      perror("Auto1 :  : gethostbyname");
      exit(-1);
      }
   /* initialisation de la structure */
   memset(&Sockaddr4,0,sizeof(struct sockaddr_in));
   /* on travaille sur TCP/IP */
   Sockaddr4.sin_family = PF_INET;
   memcpy(&(Sockaddr4.sin_addr),Machine4->h_addr,Machine4->h_length);

   /* on ecoute sur quel port ???? */
   Service4 = getservbyname(argv[5],"tcp");
   if (Service4 == NULL)
      {
      perror("Auto1 :  : getservbyname");
      exit(-2);
      }
   Sockaddr4.sin_port = Service4->s_port;

   Piece = 0;
   while (1)
      {
      printf("Auto2 : demander piece ... \n");
      res = -1;
      while (res == -1)
           {
           /* On peut avoir le socket */
           /* PF_INET => socket tcp/ip */
           /* SOCK_STREAM => tcp comme protocol */
           /* 0 => protocole par defaut sur tcp */
           Fd21 = socket(PF_INET,SOCK_STREAM,0);
           if (Fd21 == -1)
              {
              perror("Auto2 : socket");
              exit(-3);
              }
             res=connect(Fd21,(struct sockaddr *)
&Sockaddr21,sizeof(struct sockaddr_in));
             if ( res < 0)
              {
              perror("Auto2 : connect");
              close(Fd21);
              sleep(1);
              }
             }
      /* on Demande la piece ... */
      write(Fd21,MESSAGE,strlen(MESSAGE) +1 );
```

```
      read(Fd21,Reponse,TAILLEMAX +1 );
      sscanf(Reponse,"%d",&Piece);
      close(Fd21);
      printf("Auto2 : Traiter piece numero %d \n",Piece);
      sleep(5);
      /* Informer Auto4 */
     /* On peut avoir le socket */
     /* PF_INET => socket tcp/ip */
     /* SOCK_STREAM => tcp comme protocol */
     /* 0 => protocole par defaut sur tcp */
     Fd4 = socket(PF_INET,SOCK_STREAM,0);
     if (Fd4 == -1)
        {
        perror("Auto1 :  : socket");
        exit(-3);
        }
      if (connect ( Fd4,(struct sockaddr *) &Sockaddr4,sizeof(struct
sockaddr_in))== -1)
         {
         perror("Auto2 : connect");
         exit(-11);
         }
      sprintf(Message4,"Auto2 a fini la piece numero %d",Piece);
      write(Fd4,Message4,strlen(Message4)+1);
      close(Fd4);

      printf("Auto2 : Attendre Demande...\n");
      /* on accepte la connexion */
      Lenght = sizeof(struct sockaddr_in);
      if ((FdAccept23= accept(Fd23,(struct sockaddr *) &Sockaddr23,
&Lenght)) == -1)
         {
         perror("Auto2 :  : accept");
         exit (-6);
         }

      /* on tente la lecture du message */
      read(FdAccept23,Message,TAILLEMAX);
      /* On devrait testert la validiter de la demande .... */
      sprintf(Reponse,"%d :Voici la piece ",Piece);
      write(FdAccept23,Reponse,strlen(Reponse)+1);
      close(FdAccept23);
      }
   close(Fd23);
   exit(0);
   }
```

automate 3

```
/* Fichiers a inclure .... */
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/* Constantes */
/* Message envoye au serveur */
#define MESSAGE " Message du client .... "
#define TAILLEMAX 100

/* Prototypages */
int main(int argc, char ** argv);

int main(int argc, char ** argv)
   {
   int Fd; /* Descripteur de fichier (socket) */
   int Fd4; /* Descripteur de fichier (socket) */
   struct sockaddr_in Sockaddr; /* Descriptif d'un socket */
   struct sockaddr_in Sockaddr4; /* Descriptif d'un socket */
   struct hostent * Machine4; /* Machine ou reside le serveur4 */
   struct servent * Service4; /* Service (ou port) ou le serveur4 est
attache */
   struct hostent * Machine; /* Machine ou reside le serveur */
   struct servent * Service; /* Service (ou port) ou le serveur est
attache */
   int res;              /* pour tester le connect */
   char Reponse[TAILLEMAX +1];/* Reponse du serveur */
   char Message4[TAILLEMAX +1];/* Reponse du serveur */
   int Piece ; /* numero de la piece a traiter */

   /* verifier le lancement */
   if ( argc != 5)
      {
      fprintf(stderr," Auto3 : Syntaxe : %s : Machine Port Machine4
Port4...\n",argv[0]);
      exit(-10);
      }
   /* addresse de la machine du serveur */
   Machine = gethostbyname(argv[1]);
   if (Machine == NULL)
      {
      perror("Auto3 : gethostbyname");
      exit(-1);
      }
   /* initialisation de la structure */
   memset(&Sockaddr,0,sizeof(struct sockaddr_in));
   /* on travaille sur TCP/IP */
   Sockaddr.sin_family = PF_INET;
   memcpy(&Sockaddr.sin_addr,Machine->h_addr,Machine->h_length);
```

```
   /* on ecoute sur quel port ???? */
   Service = getservbyname(argv[2],"tcp");
   if (Service == NULL)
      {
      perror("Auto3 : getservbyname");
      exit(-2);
      }
   Sockaddr.sin_port = Service->s_port;
/* Initialisation vers Automat4 */
   Machine4 = gethostbyname(argv[3]);
   if (Machine4 == NULL)
      {
      perror("Auto3 :  : gethostbyname");
      exit(-1);
      }
   /* initialisation de la structure */
   memset(&Sockaddr4,0,sizeof(struct sockaddr_in));
   /* on travaille sur TCP/IP */
   Sockaddr4.sin_family = PF_INET;
   memcpy(&(Sockaddr4.sin_addr),Machine4->h_addr,Machine4->h_length);

   /* on ecoute sur quel port ???? */
   Service4 = getservbyname(argv[4],"tcp");
   if (Service4 == NULL)
      {
      perror("Auto3 :  : getservbyname");
      exit(-2);
      }
   Sockaddr4.sin_port = Service4->s_port;
   while (1)
      {
      printf("Auto3 : demander piece ... \n");
      res = -1;
      while (res == -1)
           {
           /* On peut avoir le socket */
           /* PF_INET => socket tcp/ip */
           /* SOCK_STREAM => tcp comme protocol */
           /* 0 => protocole par defaut sur tcp */
           Fd = socket(PF_INET,SOCK_STREAM,0);
           if (Fd == -1)
              {
              perror("Auto3 : socket");
              exit(-3);
              }
            res=connect(Fd,(struct sockaddr *)
&Sockaddr,sizeof(struct sockaddr_in));
            if ( res < 0)
             {
             perror("Auto3 : connect");
             close(Fd);
             sleep(1);
             }
            }
```

```
        /* on Demande la piece ... */
        write(Fd,MESSAGE,strlen(MESSAGE) +1 );
        read(Fd,Reponse,TAILLEMAX +1 );
        sscanf(Reponse,"%d",&Piece);
        close(Fd);
        printf("Auto3 : Traiter piece numero %d \n",Piece);
        sleep(5);
        /* Informer auto4 */
       /* On peut avoir le socket */
       /* PF_INET => socket tcp/ip */
       /* SOCK_STREAM => tcp comme protocol */
       /* 0 => protocole par defaut sur tcp */
       Fd4 = socket(PF_INET,SOCK_STREAM,0);
       if (Fd4 == -1)
          {
          perror("Auto3 :  : socket");
          exit(-3);
          }
        if (connect ( Fd4,(struct sockaddr *) &Sockaddr4,sizeof(struct
sockaddr_in))== -1)
           {
           perror("Auto3 : connect");
           exit(-11);
           }
        sprintf(Message4,"Auto3 a fini la piece numero %d",Piece);
        write(Fd4,Message4,strlen(Message4)+1);
        close(Fd4);

        printf("Auto3 : Enlvevez la piece puis RETURN \n");
        getchar();
        }
    exit(0);
    }
```

automate 4

```
/* Fichiers a inclure .... */
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/utsname.h>
#include <netdb.h>
#include <signal.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/* Constantes */
      /* Message envoye au serveur */
#define TAILLEMAX 100 /* Pour chaine de caracteres */


/* Prototypages */
int main(int argc, char ** argv);
void Repondeur(int Fd);

int main(int argc, char ** argv)
{
pid_t Pid;
int Fd; /* Descripteur de fichier (socket) */
int FdAccept; /* Descripteur de fichier (socket) */
struct sockaddr_in Sockaddr; /* Descriptif d'un socket */
struct hostent * Machine; /* Machine ou reside le serveur */
struct servent * Service; /* Service (ou port) ou le serveur est
attache */
struct utsname Uname;  /* Informations sur la machine (non,
version,...) */
char Message[TAILLEMAX +1];
char NomMachine[SYS_NMLN +1];
int Lenght;
struct sigaction Action; /* Gestion des signaux */

/* Verifier l'appel */
if (argc != 2)
      {
      fprintf(stderr,"Syntaxe : %s Port...\n",argv[0]);
      exit (-7);
      }

/* Obtenir le nom de la machine sur le reseau */
if ( uname(&Uname) == -1)
  {
  perror("Serveur: uname :");
  exit(-10);
  }
strcpy(NomMachine,Uname.nodename);
```

```
/* le serveur est sur la machine ou on lance le programme */
Machine = gethostbyname(NomMachine);
if (Machine == NULL)
      {
      perror("Serveur : gethostbyname");
      exit(-1);
      }
/* initialisation de la structure */
memset(&Sockaddr,0,sizeof(struct sockaddr_in));
/* on travaille sur TCP/IP */
Sockaddr.sin_family = PF_INET;
memcpy(&(Sockaddr.sin_addr),Machine->h_addr,Machine->h_length);

/* on ecoute sur quel port ???? */
Service = getservbyname(argv[1],"tcp");
if (Service == NULL)
      {
      perror("Serveur : getservbyname");
      exit(-2);
      }
Sockaddr.sin_port = Service->s_port;

/* On peut avoir le socket */
/* PF_INET => socket tcp/ip */
/* SOCK_STREAM => tcp comme protocol */
/* 0 => protocole par defaut sur tcp */
Fd = socket(PF_INET,SOCK_STREAM,0);
if (Fd == -1)
      {
      perror("Serveur : socket");
      exit(-3);
      }

/* reserver le port sur lequel on veut ecouter */
if (bind(Fd,(struct sockaddr *) & Sockaddr,sizeof(struct
sockaddr_in)) == -1)
      {
      perror("Serveur : bind");
      exit (-4);
      }

/* on attend au maximum une connection a la fois */
if (listen(Fd,10) == -1)
      {
      perror("Serveur : listen");
      exit (-5);
      }
```

```
while (1)
      {
      /* on accepte la connexion */
      Lenght = sizeof(struct sockaddr_in);
      if ((FdAccept = accept(Fd,(struct sockaddr *) &Sockaddr,
&Lenght)) == -1)
            {
            perror("Serveur : accept");
            exit (-6);
            }
      /* On ignore la mort des repondeurs */
      Action.sa_handler = SIG_IGN;
      Action.sa_flags = SA_NOCLDWAIT;
      if (sigaction(SIGCHLD,&Action,NULL) == -1)
            {
            perror("Automat4 sigaction");
            exit(-30);
            }
      /* On lance le repondeur */
      Pid = fork();
      switch(Pid)
            {
            case -1 : /* Erreur */
                  perror("Fork");
                  exit(-1);
                  break;
            case 0:     /* Le fils qui va etre le repondeur */
                  /* fermer le socket du bind */
                  close(Fd);
                  /* Apeller le repondeur */
                  Repondeur(FdAccept);
                  exit(0);
                  break;
            default : /* le pere */
                  /* rien a dire don fermer le fdaccept */
                  close(FdAccept);
                  break;
            }
      }
close(Fd);
exit(0);
}
/*****************************************/
/* Le repondeur recoit le socket de dialoque*/
/*****************************************/
void Repondeur(int Fd)
{
char Message[TAILLEMAX+1];

memset(Message,'\0',TAILLEMAX+1);
read(Fd,Message,TAILLEMAX);
printf("%s\n",Message);
close(Fd);
}
```