

00050
00100
00150
00200
00250
00300
00350
00400
00450
00500
00550
00600
00650
00700
00750
00800
00850
00900
00950
01000
01050
01100
01150
01200
01250
01300
01350
01400
01450
01500
01550
01600
01650
01700
↑L

HYDRA USER'S MANUAL

(Preliminary Version)

Ellis Cohen (Editor)
Dave Jefferson
Tom Lane
Roy Levin
Fred Pollack
Bill Wulf

Dept Computer Science
Carnegie Mellon University

Nov 1974

This is a preliminary version of the Hydra Manual
for friends. Please report any corrections
or comments to Ellis Cohen [N810EC03]@CMUA

This work was supported by the Advanced Research Projects
Agency of the Department of Defense under contract no.
F44620-73-C-0074 and monitored by the Air Force Office of
Scientific Research.

00050 .SEC |INTRODUCTION|

00100

00150 This document is a user's manual for the HYDRA Kernel. A certain
00200 amount of tutorial material can be found in the manual. Readers with a
00250 sketchy background in protection are advised to first read the HYDRA
00300 article in the CACM.

00350

00400 We want to stress strongly that HYDRA is not by itself an Operating
00450 System in the usual sense, rather it augments the PDP-11 to provide a
00500 well-protected basis on which an Operating System can be built.

00550 Hence, HYDRA is known as the KERNEL of an Operating System. In fact,
00600 many different Operating Systems can be running on HYDRA
00650 simultaneously. A standard System is available and is the one that a
00700 user initially interacts with when she logs in. This standard system
00750 is described in a separate document.

00800

00850 HYDRA provides a software virtual machine implemented on C.mmp
00900 (Carnegie Multi-Mini Processor, though "C." actually stands for
00950 "Computer"), a network of PDP-11 processors. The virtual machine
01000 instructions are known as KALLs (Kernel cALLs). They are described in
01050 terms of a standard set of BLISS-11 Macros (available on
01100 HYKALL.R11[N810HY00]). Hence, no knowledge of the PDP-11 is necessary
01150 to understand much of the contents of this manual. The Appendix
01200 contains a listing of HYKALL.R11 as well as examples of the machine
01250 code calling sequence for various KALLs.

01300

01350

↑L

00050 .SEC |THE BASIC KERNEL|

00100

00150 .SUBSEC |A CAPABILITY SYSTEM|

00200

00250 The HYDRA Kernel provides an execution environment in which
00300 protection plays a key part. In some systems, FILEs are the units of
00350 protection, in others, SEGMENTS. In HYDRA, the basis of protection is
00400 an entity called an OBJECT.

00450

00500 Many traditional operating systems are 'Access Control Systems';
00550 that is, protection information is associated with the Object being
00600 protected. For example, in the PDP-10 TOPS Operating System, when an
00650 executing procedure tries to open a file (using an ASCII encoding of
00700 the file name), the access key associated with the file is checked.

00750

00800 HYDRA, on the other hand, is a 'Capability System'. As we noted, the
00850 basis of protection in HYDRA is an entity called an OBJECT, and the
00900 protection system is invoked to determine whether particular accesses
00950 to Objects will be allowed. In a Capability System, associated with
01000 each executing Procedure is a C-List, a list of Capabilities; each
01050 Capability contains the name of an Object and a set of Rights which
01100 determine how that Object may be accessed by the executing procedure.

01150

01200 Each different Object is assigned a unique name by the Kernel.
01250 Rather than showing 'real' unique names in diagrams, (represented
01300 internally by unique 64 bit combinations), we will instead substitute
01350 unique alphanumeric names for pictorial clarity.

01400

01450 In HYDRA, Objects are Typed. Examples of Types built into HYDRA
01500 (called Kernel Types) are PAGEs, DEVICEs and PROCESSEs. There is also
01550 a facility to allow the creation of new user types. Certain types
01600 represent physical resources (e.g. Objects of Type DEVICE represent
01650 actual devices; one may represent a disk, another a line printer,
01700 etc.), but in general, Types represent abstractions of resources, both
01750 physical and virtual, and Objects of such a Type have meaning only in
01800 terms of their 'Representation' and how that representation is
01850 accessed and manipulated.

01900

01950 HYDRA is a paged system. When a procedure executes, its code (and
02000 directly accessible data) is contained in pages represented by PAGE
02050 Objects. Capabilities for these PAGE Objects must be in the C-List of
02100 the executing procedure. The Paging Section describes how to indicate
02150 to the Kernel which of these should be made directly addressable.

02200

02250 In HYDRA, an executing Procedure is a distinct type of Object,
02300 called an LNS (Local Name Space) and differs from the Type
02350 representing its static counterpart, a PROCEDURE. PROCESS Objects are
02400 the scheduling entities of the Kernel. Each running Process has an
02450 LNS associated with it which determines the 'Environment' in which the
02500 process runs. HYDRA provides a CALL Mechanism to change environments
02550 - by associating a different LNS with a process.

02600

02650

02700 .SUBSEC |OBJECTS, CAPABILITIES AND PATHS|

02750
02800
02850
02900
02950
03000
03050
03100
03150
03200
03250
03300
03350
03400
03450
03500
03550
03600
03650
03700
03750
03800
03850
03900
03950
04000
04050
04100
04150
04200
04250
04300
04350
04400
04450
04500
04550
04600
04650
04700
04750
04800
04850
04900
04950
05000
05050
05100
05150
05200
05250
05300
05350
05400

Every type of Object has two parts, a C-List containing a list of Capabilities, and a Data-Part containing data. The C-List and Data-Part of an Object together comprise its 'Representation'.

Both the C-List and Data-Part are linearly ordered, based at 1. The maximum number of Capabilities in a C-List and the maximum length of a Data-Part varies from type to type. The Appendix contains those numbers for Kernel types. Since C-Lists are linearly ordered, we will often refer to a Capability as being in the k'th 'Slot' of a C-List.

As examples, consider the representation of some Kernel Objects: A PAGE Object contains an empty C-List and its Data-Part contains the location of the page (Disk, Drum or Core address) and its status. The Data-Part of a Device Object contains a code identifying the device. The Data-Part of an LNS contains (among other things) trap addresses, a mask of processors on which the LNS may execute, and paging information, while the C-List of the LNS contains the Capabilities which define the 'Environment' of the LNS.

There are facilities for creating new Types of Objects as well as for creating Objects of existing types and erasing them. For example, a user might create a new Type of Object, a FILE, whose C-List might contain Capabilities for PAGEs and whose Data-Part might contain information about the file (it could even be used to hold access keys as part of a system that could provide file access checking in a way similar to that of the PDP-10 TOPS monitor). Or a user might create a DIRECTORY Type. Objects of type DIRECTORY might have a C-List containing Capabilities for FILEs and other DIRECTORYs. This could be used to build up an hierarchical FILE system similar to the one in MULTICS.

C-Lists and Data-Parts can only be accessed and manipulated through the Kernel via KALLs. The Kernel provides some very basic Kalls that do the following kinds of things: Delete Capabilities from the C-List of some Object, Move a Capability from the C-List of one Object to the C-List of another Object (perhaps the same) (with or without deleting the first Capability) and move data to and fro between the Data-Part of some Object and directly addressable memory. Of course, we again stress that these operations cannot be performed on arbitrary objects, rather, the executing LNS must have a Capability for the Object to be accessed.

Most KALLs require some arguments which specify Capabilities. In the simplest case, these are denoted by SIMPLE INDEXes into the C-List of the LNS. For example, there is a KALL, 'DELETE', and DELETE (3) Kalls the Kernel to eliminate the 3rd Capability in the LNS executing that KALL. Often, the Kernel will allow a Capability to be denoted by a PATH INDEX (See Diagram 2). For example, DELETE (Path(3,4,2,1)) will delete the 1st Capability in the Object referenced by the 2nd Capability in the the Object referenced by the 4th Capability in the Object referenced by the 3rd Capability in the executing LNS. The Capability deleted is called the TARGET of Path(3,4,2,1). The Capability denoted by Path(3,4,2) is called the PRETARGET and the

05450 Capabilities denoted by Path(3,4) and 3 are called STEPS. (Note: the
05500 denotation Path(3) is the same as just 3; such paths are called
05550 Simple)

05600
05650 .SUBSEC |KERNEL RIGHTS AND RIGHTS RESTRICTION|
05700

05750 As we noted, HYDRA implements basic protection through a set of
05800 rights. The right to perform some class of accesses (via KALLs of
05850 course) with respect to a Capability is determined by the presence of
05900 a particular bit in the Rights field of a Capability. (For a listing
05950 of all rights and respective bits, see the Appendix) The following is
06000 a description of the rights relevant to basic The following is a
06050 description of the rights relevant for basic Kernel Kalls. In
06100 describing these rights, we consider the effect of Capability CAP
06150 having the right in question. If CAP is an Object Reference, we write
06200 OBJ as a shorthand for the Object Referenced by CAP.

06250
06300 Capability Rights

06350
06400 DLTRTS - Allows CAP to be Deleted

06450
06500 ENVRTS - Allows CAP to be Stored in some Object

06550
06600 C-List Rights

06650
06700 LOADRTS - Allows a Capability to be Loaded from OBJ's C-List

06750
06800 STORTS - Allows a Capability to be Stored into OBJ's C-List

06850
06900 APPRTS - Allows a Capability to be Appended onto OBJ's C-List

06950
07000 KILLRTS - Allows a Capability to be Deleted from OBJ's C-List

07050
07100 Data-Part Rights

07150
07200 GETRTS - Allows data to be gotten from OBJ's Data-Part

07250
07300 PUTRTS - Allows data to put into OBJ's Data-Part

07350
07400 ADDRRTS - Allows data to be appended onto OBJ's Data-Part

07450
07500 Restriction Rights

07550
07600 MDFYRTS - Allows modification of either OBJ's C-List or Data-Part

07650
07700 UCNFRTS - Allows OBJ to be 'UnCoNFined', that is, an Object
07750 accessed through OBJ may be modified.

07800
07850
07900 Some examples:

07950
08000 DELETE (3) (The Capability denoted by) 3 requires DLTRTS

08050
08100 DELETE (Path(3,4)) 3 requires KILLRTS & MDFYRTS,

08150 Path(3,4) requires DLTRTS
08200
08250 DELETE (Path(3,4,2,1)) 3 and Path(3,4) require LOADRTS & UCNFRTS
**,
08300 Path(3,4,2) requires KILLRTS & MDFYRTS,
08350 Path(3,4,2,1) requires DLTRTS
08400
08450 LOAD(x,y) is a KALL which moves the Capability at y to x, retaining
08500 the Capability at y. x must be a Simple Index.
08550
08600 LOAD (4, Path(3,4,2)) 3 requires LOADRTS
08650 Path(3,4) requires LOADRTS
08700 4 must be an empty slot
08750
08800 Note that when a Capability is moved, it picks up DLTRTS, while
08850 the other rights remain the same as in the original.
08900
08950 TAKE(x,y) is just like LOAD but also deletes the Capability at y.
09000
09050 TAKE (5, Path(3,4,3)) 3 requires LOADRTS & UCNFRTS
09100 Path(3,4) requires LOADRTS,
09150 MDFYRTS & KILLRTS
09200 Path(3,4,3) requires DLTRTS
09250 5 must be an empty slot
09300
09350 There is often a desire to restrict the Rights of a Capability
09400 when it is copied from one's own LNS to the C-List
09450 of another Object. Hence, the Kall, STORE(x,y,a)
09500 moves the Capability at y to x (y must be a Simple Index), and
09550 then restricts the rights of the Capability at x according to
09600 the contents of a mask at address a (See the Appendix for
09650 the format), by eliminating those rights not represented by a 1 in
09700 the mask.
09750
09800 STORE (Path(3,4,3), 2, addr) 3 requires LOADRTS & UCNFRTS
09850 Path(3,4) requires STORTS & MDFYRTS
09900 Path(3,4,3) must be an empty slot
09950 2 requires ENVRTS
09955
09960 If the address designating the rights restriction mask is zero,
09965 no rights are restricted. If the address is non-zero, then ALLYRTS
09970 (described in a later section) are always restricted regardless
09975 of whether the mask indicates that they should be.
10000
10050
10100 .SUBSEC [AUXILIARY RIGHTS AND KERNEL TYPES]
10150
10200 The Rights we have seen so far are called Kernel Rights because they
10250 have meaning for any Capability regardless of the Type of the Object
10300 it references. In addition, each Capability also contains a field of
10350 Auxiliary rights that may be defined differently for each new Type of
10400 Object. Their use will become apparent in future examples.
10450
10500 The Kernel recognizes a basic set of Types and treats them

10550 seperately. Their auxiliary rights have predefined meanings and the
10600 Kernel also limits the Kernel rights that any Capability for an Object
10650 of one of these Types may have.

10700
10750 .SUBSEC |TYPES NULL, DATA & UNIVERSAL|
10800

10850 Objects of Type NULL represent absolutely nothing. They are
10900 constrained by the Kernel to have neither a C-List nor a Data-Part.
10950 What we have thus far referred to as an 'Empty slot' in a C-List
11000 contains a NULL Capability. The 'Length' of a C-List is the index of
11050 the last non-Null in the C-List. A Capability slot is said to be
11100 'Defined' if its index is not greater than the Length of the C-List it
11150 refers to. In actuality, the preceding is a bit of a simplification.
11200 More details can be found in the Subsection on Nulls Revisited.

11250
11300 It is often convenient to be able to create a new Object which
11350 simply encapsulates some data. The Kernel provides a Kall, 'DATA'
11400 which does the encapsulation, creating a new Object of Type DATA whose
11450 Data-Part contains the data. DATA Objects have no C-List and have no
11500 defined Auxiliary rights.

11550
11600 It is also convenient to provide a UNIVERSAL Object, one with both a
11650 C-List and a Data-Part. The Kall UNIV creates just such an Object.

11700
11750
11800 .SUBSEC |KALL VALUES AND SIGNALS|
11850

11900 Any KALL that executes successfully returns a non-negative value in
11950 register R\$0. KALLs that fail (e.g. inadequate rights) return a
12000 negative value, called a "Signal" (In addition, certain additional
12050 signal related information is sometimes placed in SIGDATA, a fixed
12100 location in the stack page). There is also a mechanism that can
12150 force signals to cause user traps (See the section on Procedure & LNS
12200 Context Blocks for more details). A complete listing of signals and
12250 their values can be found in the Appendix. The meaning of the various
12300 signals that can occur during basic Kernel KALLs can be found in the
12350 Appendix.

12400
12450
12500 .SUBSEC |LOCKING OF OBJECTS|
12550

12600 Since it is possible for two separate LNS's to contain Capabilities
12650 for the same Object, it is possible that both will be running
12700 simultaneously (on different processors) and will try to STORE
12750 different Capabilities in the same C-List slot of the shared Object.
12800 Such operations are performed indivisibly; when a Capability or Data
12850 is being moved either to or from an Object, that Object will (in
12900 general) be LOCKED. Hence, in the motivating example above, one LNS
12950 (nondeterministically will gain access to the Object and STORE a
13000 Capability in it, while the other waits on the Lock. When the STORE
13050 Kall completes, the other LNS will gain access to the Object, but its
13100 STORE Kall will fail (signal), since the slot in the shared Object
13150 will no longer be Empty.
13200

13250 For certain Kalls, if some referenced Object cannot immediately be
13300 locked, the Kall will fail. To do otherwise in those cases would
13350 allow the possibility of deadlock. For the same reason, any Kall that
13400 accesses a PROCEDURE Object (except when an LNS is being incarnated
13450 from it) must be able to lock the Procedure immediately or else the
13500 Kall will fail.

13550

13600

13650 .SUBSEC |MEMORY ADDRESSES & THE STACK|

13700

13750 PDP-11's as modified for C.mmp have a 16 bit address space and a
13800 paged architecture. Pages are 8192 bytes long. The lower 13 bits of
13850 a 16 bit address designates a byte within a page. The high order 3
13900 bits select one of 8 pages that may be directly addressable at any
13950 given time. Page 0 is designated the Stack Page to be used in
14000 conjunction with the PDP-11 SP register and is treated somewhat
14050 specially by the Kernel. HYDRA contains various KALLs that allow the
14100 user to change other pages (virtual overlaying). More details can be
14150 found in the section on PAGING. More details on the C.mmp hardware may
14200 be found in a separate document.

14250

14300 Many KALLs require one or more arguments to be memory addresses.
14350 Such memory address is expected to be the origin (low order address)
14400 of a block of memory from which the Kernel will either store or
14450 retrieve information. The KERNEL demands that these 'Legitimate Stack
14500 Memory addresses' have the following properties:

14550

14600 1) Such addresses be in the stack page (high order 3 bits of the
14650 address must be 0)

14700

14750 2) The block of memory to be accessed must lie within the active
14800 region of the stack or within the Process Communication Area,
14850 locations 0 - #176. (When an LNS begins execution, SP, the stack
14900 register, is set to point to an initial stack location. The modified
14950 PDP-11 hardware insures that SP can never be set higher than this
15000 initial value, that is the stack grows down. The region between the
15050 initial SP contents and the current contents of SP is called the
15100 Active Region of the stack).

15150

15200 3) The address must be on a word boundary (low order bit 0)

15250

15300 The stack may also be directly accessed using PDP-11 instructions
15350 since the stack is page 0. The modified C.mmp hardware prevents
15400 accesses to page 0 above the LNS's initial stack location, however,
15450 any access below that is allowed.

15500

15550 Locations 0 - #377 have special uses. Locations 0 - #177 comprise
15600 the Process Communication Area. It can be accessed by all LNS's that
15650 execute within a particular Process. Locations #200 - #377 comprise
15700 the Kernel Data Area. When signals, traps and errors occur, certain
15750 additional information is placed in locations within this area (The
15800 Appendix lists these fields) The Kernel also uses part of this area
15850 as working storage during Kalls.

15900

15950
16000
16050
16100
16150
16200
16250
16300
16350
16400
16450
16500
16550
16600
16650
16700
16750
16800
16850
16900
16950
17000
17050
17100
17150
17200
17250
17300
17350
17400
17450
17500
17550
17600
17650
17700
17750
17800
17850
17900
17950
18000
18050
18100
18150
18200
18250
18300
18350
18400
18450
18500
18550
18600

.SUBSEC |INDIRECT KALLS|

Often it is useful to be able to build up the argument stack for a KALL independently of the actual KALL itself (especially for interpretive and debugging programs). The Appendix contains all details necessary for constructing the argument stack.

The special KALL, INDKALL (Mem), where Mem is the beginning address of the argument stack and must be a Legitimate Stack Memory Address provides this function.

.SUBSEC |CONVENTIONS FOR KALL SPECIFICATIONS|

A) KALLs are described in terms of Bliss Macros. See the Appendix.

B) The 'Parameters' section. Parameters to KALLs fall into three classes.

1) An integer value

2) A Legitimate Stack Memory Address - in the sense of the Subsection on Stack Memory Addresses. Where a memory address is optional, its absence is denoted by 0.

The block of memory will in general be used either in conjunction with movement of data to or from a Data-Part or rights restriction. See the Subsection on Kernel Types and Rights Restriction and the Appendix)

3) A Denotation for a Capability - either a Simple index, (sometimes negated or 0 for a special effect) or a Path index, or a Call Parameter (to be defined in the Intermediate Kernel section). We will also indicate necessary rights, type or kind (Object Reference or Template) for the target Capability and its pretarget.

Unless we note otherwise in the specifications, we require that each STEP in a Path (Capabilities in the Path other than the Target or Pretarget) be an Object Reference Capability with LOADRTS.

We will not list restrictions on arguments that seem obvious or redundant and produce obvious signals if the restrictions are not met - most notably, indexes into C-Lists or Data-Parts less than 1 or greater than the maximum length.

C) 'Effect' is the effect of the Kall if no signal occurred. Except for two small subcases (of LNS incarnation and Page Set initialization), Kalls that fail have no side effects.

D) 'Signals' indicate unusual signals that may occur. Signals that indicate bad arguments or arguments that denote capabilities of the wrong kind or type or having inadequate rights are not mentioned. These are a possibility in almost every KALL and are described in the

18650 section on Signals above.
18700
18750 E) 'Result' is the value of the Kall (returned in R\$0) assuming no
18800 signal occurred. (If a signal occurred, the value of the Kall is the
18850 signal value instead)
18900
18950
19000 .SUBSEC |SPECIFICATIONS FOR BASIC KERNEL KALLS|
19050
19100
19150
19200 INFORMATIONAL KALLs
19250
19300
19350 GETCLOCK (Mem)
19400 Parameters:
19450 Mem - Legitimate Stack Memory address
19500 - The current LNS must not be Blind (See next section)
19550 Effect: Puts a reading of the system clock into the 4 word
19600 block of memory beginning at Mem. See the Appendix for
19650 the format.
19700 Signals:
19750 SBLND - Current LNS is Blind
19800 Result: 0
19850
19900
19950 LENTH
20000 Parameters: None
20050 Effect: None
20100 Result: Length of the C-List of the Executing LNS
20150
20200
20250 CLENTH (Path)
20300 Parameters:
20350 Path - Path index; Pretarget: LOADRTS;
20400 Target: Object Reference, LOADRTS
20450 Effect: None
20500 Result: Length of the C-List of the Object Referenced by
20550 Path's Target.
20600
20650
20700 DLENTH (Path)
20750 Parameters:
20800 Path - Path index; Pretarget: LOADRTS;
20850 Target: Object Reference, GETRTS
20900 Effect: None
20950 Result: Size of the Data-Part of the Object Referenced by
21000 Path's Target.
21050
21100
21150 WHAT (Memd, Path)
21200 Parameters:
21250 Memd - Legitimate Stack Memory address
21300 Path - Path index; Pretarget: LOADRTS; Target: Defined

21350 - The current LNS must not be Blind (See next section)
 21400 Effect: Information about the Capability targeted by Path
 21450 is stored in the 16 word block of memory beginning at Memd.
 21500 See the Appendix for the format.
 21550 Signals:
 21600 SBLND - Current LNS is Blind
 21650 Result: 0
 21700
 21750
 21800 COMPAR (Path, Ncur)
 21850 Parameters:
 21900 Path - Path index; Pretarget: LOADRTS; Target: Defined
 21950 Ncur - Simple index, Defined or 0
 22000 Effect: None
 22050 Result: A word of bits which indicate how the Capabilities
 22100 targeted by Path and Ncur compare. If Ncur is 0,
 22150 then just those bits pertaining to the Capability targeted by
 22200 Path are set. See the Appendix for the meanings of each bit.
 22250
 22300
 22350
 22400 SIMPLE DATA & UNIVERSAL MANIPULATION
 22450
 22500
 22550 GETDATA (Memd, Path, Disp, Knt)
 22600 Parameters:
 22650 Memd - Legitimate Stack Memory address
 22700 Path - Path index; Pretarget: LOADRTS; Target: GETRTS
 22750 Disp - Positive integer less than or equal to Dlenh(Path)
 22800 Knt - Positive integer
 22850 Effect: Moves up to Knt words of data from the Data-Part of
 22900 the Object referenced by the Target to the block of
 22950 memory beginning at Memd. The data is copied beginning at
 23000 the Disp'th word of the Data-Part and continuing for a
 23050 total of Knt words or until the end of the Data-Part is
 23100 reached.
 23150 Result: Total number of words copied
 23200
 23250
 23300 PUTDATA (Path, Memd, Disp, Knt)
 23350 Parameters:
 23400 Memd - Legitimate Stack Memory address
 23450 Path - Path index; Steps & Pretarget: LOADRTS,UCNFRTS;
 23500 Target: PUTRTS,MDFYRTS
 23550 Disp - Positive integer
 23600 Knt - Positive integer
 23650 Effect: Copies Knt words of data beginning at Memd into the
 23700 Data-Part of the Object targeted by Path. The data is
 23750 stored beginning at the Disp'th word of the Data-Part.
 23800 Result: 0
 23850
 23900
 23950 DATA (Path, Memd, Knt, Memr)
 24000 Parameters:

24050 Path - Path index; Steps: LOADRTS,UCNFRTS;
24100 Pretarget: STORTS,MDFYRTS; Target: Empty
24150 Memd - Legitimate Stack Memory address
24200 Knt - Non-negative integer
24250 Memr - Legitimate Stack Memory address
24300 Effect: Creates a Data Object and places a Capability for
24350 it in Path's Target. The Data-Part of the created Object
24400 will contain the Knt words of data copied from the block of
24450 memory beginning at Memd. The Capability will have all relevant
24500 rights except ALLYRTS & FRZRTS and will be further restricted
24550 by the contents of Memr if Memr is non-zero.
24600 Result: 0
24650
24700

ADDATA (Path, Memd, Knt)

Parameters:

24850 Path - Path index; Steps & Pretarget: LOADRTS,UCNFRTS;
24900 Target: ADDRRTS,MDFYRTS
24950 Memd - Legitimate Stack Memory address
25000 Knt - Positive integer

25050 Effect: Copies the Knt words of data from the block of memory
25100 beginning at Memd onto the end of the Data-Part of the
25150 Object referenced by Path's Target.

25200 Result: 0
25250
25300

UNIV (Path)

Parameters:

25450 Path - Path index; Steps: UCNFRTS,LOADRTS;
25500 Pretarget: STORTS,MDFYRTS; Target: Empty

25550 Effect: Creates a Universal Object and places a Capability for
25600 it with all but ALLYRTS & FRZRTS in Path's Target.

25650 Result: 0
25700
25750
25800

SIMPLE MANIPULATION OF CAPABILITIES

PASS (Path, Ncur, Memr)

Parameters:

26100 Path - Path index; Steps: LOADRTS,UCNFRTS;
26150 Pretarget: STORTS,MDFYRTS; Target: Empty
26200 Ncur - Simple index, DLTRTS; if Path is not Simple,
26250 requires ENVRTS as well
26300 Memr - Legitimate Stack Memory address or 0

26350 Effect: Copies the Capability in the Ncur'th slot of the current
26400 LNS to Path's target, restricting rights (if Memr
26450 is nonzero) according to the contents of Memr. Then, the
26500 Capability at Ncur is deleted.

26550 Result: 0
26600
26650

TAKE (Nnew, Path)

26700

26750 Parameters:
 26800 Nnew - Simple index, Empty
 26850 Path - Path index; Steps: LOADRTS,UCNFRTS;
 26900 Pretarget: KILLRTS,LOADRTS,MDFYRTS; Target: DLTRTS
 26950 Effect: Copies the Capability targeted by Path to the Nnew'th
 27000 slot of the current LNS. If Pretarget lacks UCNFRTS, then
 27050 Nnew will lack UCNFRTS, MDFYRTS & ALLYRTS. Then deletes the
 27100 Capability targeted by Path.
 27150 Result: 0
 27200
 27250
 27300 STORE (Path, Ncur, Memr)
 27350 Parameters:
 27400 Path - Path index; Steps: UCNFRTS,LOADRTS;
 27450 Pretarget: MDFYRTS,STORTS; Target: Empty
 27500 Ncur - Simple index, Defined; If Path is not Simple,
 27550 requires ENVRTS as well.
 27600 If Path and Ncur are the same, then none of the above Rights
 27650 requirements holds, rather the Capability needs DLTRTS.
 27700 Memr - Legitimate Stack Memory address or 0.
 27750 Effect: Copies the Capability in the Ncur'th slot of
 27800 the current LNS to Path's target, setting DLTRTS, and (if Memr
 27850 is nonzero) restricting rights according to the contents on Memr.
 27900 If Path and Ncur are the same, however, the rights
 27950 in the target are simply restricted according to the
 28000 contents of Memr (if Memr is nonzero).
 28050 Result: 0
 28100
 28150
 28200 LOAD (Nnew, Path)
 28250 Parameters:
 28300 Nnew - Simple index, Empty
 28350 Path - Path index; Pretarget: LOADRTS; Target: Defined
 28400 Effect: Copies the Capability targeted by Path
 28450 to the Nnew'th slot of the current LNS,
 28500 and sets DLTRTS. If any Capability in Target's Path lacks
 28550 UCNFRTS, Nnew will have UCNFRTS, MDFYRTS & ALLYRTS removed.
 28600 Result: 0
 28650
 28700
 28750 PASSAPPEND (Path, Ncur, Memr)
 28800 Parameters:
 28850 Path - Path index; Steps & Pretarget: LOADRTS,UCNFRTS;
 28900 Target: MDFYRTS,APPRTS
 28950 Ncur - Simple index, DLTRTS,ENVRTS
 29000 Memr - Legitimate Stack Memory address or 0
 29050 Effect: Appends the Capability in the Ncur'th slot of the current
 29100 LNS onto the end of the C-List of the Object referenced
 29150 by Path's target, restricting rights (if Memr is nonzero)
 29200 according to the contents of Memr. Then, the
 29250 Capability at Ncur is deleted.
 29300 Result: 0
 29350
 29400 APPEND (Path, Ncur, Memr)

29450 Parameters:
 29500 Path - Path index; Steps & Pretarget: UCNFRTS,LOADRTS;
 29550 Target: MDFYRTS,APPRTS
 29600 Ncur - Simple index, ENVRTS
 29650 Memr - Legitimate Stack Memory address or 0
 29700 Effect: Appends the Capability in the Ncur'th slot of the current
 29750 LNS onto the end of the C-List of the Object referenced
 29800 by Path's target, setting DLTRTS, and restricting rights
 29850 (if Memr is nonzero) according to the contents of Memr.
 29900 Result: 0
 29950
 30000
 30050
 30100 DELETE (Path)
 30150 Parameters:
 30200 Path - Path index; Steps: UCNFRTS,LOADRTS;
 30250 Pretarget: MDFYRTS,KILLRTS; Target: DLTRTS
 30300 Effect: Deletes the Capability targeted by Path. See
 30350 the Section on Types, Creating & Erasing in
 30400 the next section for other potential effects.
 30450 Result: 0
 30500
 30550
 30600 INTERCHANGE (Path, Ncur, Memr)
 30650 Parameters:
 30700 Path - Path index; Steps: UCNFRTS,LOADRTS
 30750 Pretarget: MDFYRTS,KILLRTS,LOADRTS,STORTS;
 30800 Target: DLTRTS
 30850 Ncur - Simple Index, DLTRTS
 30900 Memr - Legitimate Stack Memory address or 0
 30950 Effect: Interchanges the Capabilities targeted by Path and by Ncur.
 31000 Restricts rights (if Memr is nonzero) of the Capability
 31050 placed into Path's target according to the contents of Memr.
 31100 If Pretarget lacks UCNFRTS, Ncur will have UCNFRTS, MDFYRTS &
 31150 ALLYRTS removed.
 31200 Result: 0
 31250
 31300
 31350
 ↑L

00050 .SEC |THE INTERMEDIATE KERNEL|

00100

00150 .SUBSEC |DOMAIN SWITCHING|

00200

00250 When an executing program wishes to invoke another program (e.g.
00300 call a subroutine), the caller may not trust the called program and
00350 may wish to isolate it in a separate environment (LNS), specifying as
00400 arguments only Capabilities for those Objects in its own LNS that it
00450 wishes the called program to be able to access. Alternatively, a
00500 program that manipulates a data base needs Capabilities to access the
00550 data base but it should never be necessary for callers of the program
00600 to have direct access to the data base.

00650

00700 To solve both problems, HYDRA provides PROCEDURE Objects. The Call
00750 CALL(Rtrn,Proc,A1,...,Ak) creates a new LNS in which the Procedure's
00800 code will execute and transfers control to it. (Proc denotes a
00850 Capability for a Procedure Object, A1 through Ak denote Capabilities
00900 to be passed as arguments to the called procedure and Rtrn denotes a
00950 slot where the called Procedure may return a Capability) The Call
01000 KRETURN passes control back to the calling LNS, optionally returning a
01050 Capability.

01100

01150 The C-List of a PROCEDURE contains Capabilities that will be
01200 duplicated in each LNS incarnated from the PROCEDURE (these are called
01250 inherited Capabilities and can be used to solve the Data Base problem
01300 mentioned just above). In addition, some of the Capabilities in the
01350 Procedure's C-List are Parameter Templates. Capabilities passed as
01400 arguments to the Procedure will appear in those slots in the LNS's
01450 C-List where Parameter Templates appeared in the Procedure's C-List.
01500 In addition to specifying where Call arguments appear in the
01550 incarnated LNS, Parameter Templates also specify a type and
01600 check-rights. A Call will fail (signal) if some argument is not of
01650 the same type and does not contain the minimum rights specified by the
01700 corresponding Parameter Template.

01750

01800 It is often useful to build 'Protected Subsystems'. Consider a
01850 Directory system where users have Capabilities for directories they
01900 can access, but because the 'Directory Subsystem' maintains the
01950 directories in a special private format, users should not be able to
02000 directly access or manipulate their directories except through
02050 PROCEDURES which comprise the 'Directory Subsystem'. HYDRA
02100 accomplishes this through 'Rights Amplification'. Capabilities passed
02150 as arguments in a CALL need not have the same rights in the incarnated
02200 LNS as in the LNS of the CALLER. The Parameter Template may specify
02250 new rights which may be greater than the rights of the Capability
02300 passed as an argument; in the incarnated LNS, the Capability will have
02350 these new amplified rights.

02400

02450 The diagram notes how this solves the Directory problem through the
02500 use of auxiliary rights and parameter templates which specify
02550 new-rights. The user's Capability for a Directory does not contain
02600 rights which allow manipulation or access to the directories directly.
02650 Rather various procedures of the 'Directory Subsystem' have parameter
02700 templates which specify these rights as new-rights, so that

02750 manipulation or access of a directory can only take place in the
02800 protected environment of the 'Directory Subsystem'. Note how
02850 auxiliary rights are used to control how a Directory may be used.
02900 Since different procedures specify different check-rights for
02950 Directories passed as arguments, auxiliary rights provide a way of
03000 specifying procedural protection. HYDRA does not permit parameter
03050 Templates which specify new-rights to be created anywhere, otherwise
03100 the protection afforded by the directory system could be easily
03150 circumvented. Templates which specify new-rights can only be created
03200 using special Capabilities (See the Subsection on Types, Creating &
03250 Erasing), and since Templates are Capabilities, their dispersion can
03300 be controlled. In the above case, the presumption is that only
03350 PROCEDURES of the 'Directory Subsystem' would have Parameter Templates
03400 of Directory Type with New-Rights.
03450

03500 Creation of an LNS and transfer of control to its code can be
03550 separated. The Kall MAKLNS incarnates an LNS from a Procedure and
03600 arguments, while the Kall LNSCALL transfers control to the LNS. The
03650 advantage of having such 'Canned' LNS's is efficiency as well as the
03700 ability to build coroutine structures. Once an LNS KRETURNS, it may
03750 be LNSCALLED again. Execution continues after the KRETURN. The LNS's
03800 pages, its C-List and registers R\$0 and the PC will be retained,
03850 however, the rest of the registers will be clobbered and the stack
03900 will be reinitialized.
03950

04000

04050 .SUBSEC |TEMPLATES & MERGING|

04100

04150 The process of comparing a Capability to a Template and producing a
04200 new Capability is called 'Merging'. It is useful not only as part of
04250 the Call Mechanism, but at other times as well. Hence, there are
04300 Capability Templates (for general merging) as well as Parameter
04350 Templates (for Call-time merging). Templates contain 2 flags.
04400

04450 TEMPLFLAG - 1 - Capability Template
04500 0 - Parameter Template

04550

04600 NEWFLAG - 1 - Amplify rights in Merging (new-rights)
04650 0 - No amplification
04700

04750 These flags, if set, may be cleared in exactly the same way that
04800 rights may be restricted. Once cleared, they may not be set again.
04850 Since unlike Object References, Templates do not refer to specific
04900 Objects, there is little need for Templates to have rights. Therefore,
04950 without much conflict, rights and new-rights have been combined. Even
05000 when new-rights are specified, there are certain rights that cannot be
05050 amplified. This is true of the Kernel rights ENVRTS, UCNFRTS, FRZRTS
05100 and ALLYRTS. They will be the same in the merged Capability as in the
05150 original regardless of amplification.
05200

05250

05300

05350 .SUBSEC |NULLS REVISITED|

05400

'Empty slots' have already been defined as slots containing NULL

05450 Capabilities. In fact, it is impossible to create a NULL Object, and
05500 empty slots contain NULL Templates.

05550
05600 NULLs have one auxiliary right predefined, NULLRTS. We use the
05650 term 'Truenuil' to mean a Null Template with both NULLRTS and TEMPLFLAG
05700 set. When an Object is initially created, its C-List is set to
05750 contain all 'Truenuils' with all Kernel rights. A deleted Capability
05800 is also replaced by a Truenuil.

05850
05900 The 'Length' of a C-List is really the index of the last
05950 non-Truenuil in the C-List. Hence NULL Parameter Templates or NULL
06000 Templates lacking NULLRTS are included in the Length.

06050

06100

06150 .SUBSEC |CONFINEMENT, FREEZING, BLINDNESS & REVOCATION|

06200

06250 A number of Kernel rights are provided to solve some interesting
06300 protection problems. ENVRTS, MDFYRTS & UCNFRTS are all used to solve
06350 variants of the 'Confinement Problem'. That is, they may be used to
06400 guarantee that Capabilities and data do not escape from particular
06450 LNS's; those LNS's are then said to be confined or partially confined
06500 with respect to the information whose leakage we wish to protect
06550 against.

06600

06650 ENVRTS can be used to guarantee that Capabilities are not stored by
06700 a Callee who is passed the Capability. Without ENVRTS, the Capability
06750 cannot be placed in the C-List of any Object. It may be used as an
06800 argument to an LNS which the Callee Calls, but ENVRTS cannot be gained
06850 through rights amplification.

06900

06950 As an example, Capabilities for LNS's never have ENVRTS and thus can
07000 never be accessed or manipulated outside of the Process in which the
07050 LNS has been incarnated.

07100

07150 MDFYRTS and UCNFRTS can be used to protect Objects from modification
07200 through Capabilities lacking those rights. If an LNS calls another
07250 LNS passing a Capability lacking MDFYRTS, that guarantees that the
07300 Callee cannot modify the accessed Object through that Capability
07350 regardless of amplification. This is because MDFYRTS cannot be gained
07400 through rights amplification and any Call that modifies an Object
07450 requires a Capability for that Object with MDFYRTS as well as other
07500 relevant rights.

07550

07600 UCNFRTS also cannot be gained through amplification and prevents
07650 modification of any Object reached through the C-List of an Object
07700 referenced through a Capability lacking UCNFRTS.

07750

07800 Users may wish to guarantee that information passed to an untrusted
07850 procedure will not be leaked to another user. The Kernel right UCNFRTS
07900 also provides this guarantee. Any LNS incarnated from a Procedure
07950 Capability lacking UCNFRTS will be 'Confined'. Each Capability in the
08000 LNS inherited from the Called Procedure will lose UCNFRTS & MDFYRTS.
08050 Confinement is then provided in the following way. The reader may
08100 note that any Call which modifies an Object requires that the

08150 Capability for the Object have MDFYRTS and that other Capabilities in
08200 the Path to the Object have UCNFRTS. Additionally, whenever a
08250 Capability is loaded into an LNS through a Path where some Capability
08300 lacks UCNFRTS, the loaded Capability will have UCNFRTS, MDFYRTS and
08350 ALLYRTS removed. Hence, information and Capabilities cannot be stored
08400 by a Confined LNS through any Capabilities except those passed as
08450 parameters in incarnating the LNS.
08500

08550 Note that if a Procedure Capability with UCNFRTS is used as an
08600 argument in incarnating a Confined LNS, the Confined LNS will be able
08650 to Call an Unconfined LNS through it. Otherwise, since all inherited
08700 Capabilities of the Confined LNS lack UCNFRTS, any LNS called will be
08750 Confined as well.
08800

08850 There are still a small number of ways to covertly leak a few bits
08900 of information out of a confined LNS. It would be counterproductive
08950 to list these. However, no large leakage of data is possible.
09000

09050 Users may also wish to guarantee that an Object they have access to
09100 is 'Frozen', that is, the Object and all Objects reached by taking a
09150 Path through it will NEVER be modified, even by concurrently executing
09200 LNS's that may have a Capability for the same Object. The right
09250 FRZRTS is used like a flag to guarantee that an Object is frozen. The
09300 Kall FREEZE effectively freezes an Object by setting FRZRTS and
09350 eliminating UCNFRTS & MDFYRTS in what must be the only extant
09400 Capability for the Object. Since UCNFRTS & MDFYRTS cannot be gained
09450 through amplification, all Capabilities for the Object will lack them,
09500 guaranteeing that the Object will never be modified once frozen.
09550 FREEZE only succeeds if all Capabilities in the Object's C-List are
09600 already Frozen. So that FRZRTS can represent a guarantee of
09650 Frozen-ness, it also cannot be gained through amplification.
09700

09750 Users might further like LNS's to run 'Blind'. That is, no external
09800 information can be made available to it (the clock, process related
09850 information and other things that might change in different
09900 executions). FRZRTS also provides that function. Any LNS incarnated
09950 from a Procedure Capability with FRZRTS will be made Blind. In
10000 addition, an LNS incarnated by a Blind executing LNS will be Blind
10050 unless it is incarnated from a Procedure Capability with UCNFRTS.
10100

10150 Note that if a Procedure Capability with UCNFRTS is used as an
10200 argument in incarnating a Blind LNS, the Blind LNS will be able to
10250 Call an Unblind LNS through it. Otherwise, since all inherited
10300 Capabilities of the Blind LNS must have FRZRTS and thus must lack
10350 UCNFRTS, any LNS called will be Blind as well. Thus, with suitable
10400 arguments, execution of two Blind LNS's incarnated from the same
10450 Frozen Procedure Capability will be indistinguishable.
10500

10550 HYDRA allows Objects to act as Aliases for other Objects. Accessing
10600 such an Alias-ing Object actually causes access of the aliased Object.
10650 Aliases themselves may have aliases, allowing up to 23 levels of
10700 indirection. The Object finally accessed at the end of the alias
10750 indirection chain is called the 'Terminal Object' of an Alias.
10800

10850 An Alias may be created for any Object, and a Capability will be
10900 provided for the Alias-ing Object with ALLYRTS. With ALLYRTS, the
10950 ALiasing Object may be RE-ALLYed to act as Alias for a different
11000 Object or even for no Object at all. Thus, if a user wishes to share a
11050 Capability for an Object with another user, but might want to revoke
11100 the Capability at some later time, he need simply create an Alias for
11150 the Object and share the Capability for the Alias.

11155
11160 To guarantee that RE-ALLYing cannot be used to illicitly
11165 gain rights, whenever rights are restricted in a Capability,
11170 ALLYRTS are removed as well.

11200

11250

11300 .SUBSEC |TYPES, CREATING & ERASING|

11350

11400 Objects of Type TYPE represent all Objects in the equivalence class
11450 of a given type. For example, the Object whose name is PROCEDURE and
11500 whose Type is TYPE represents all Objects whose type is PROCEDURE.
11550 Objects of Type TYPE are used to generate Templates of the Type named
11600 by the TYPE Object. A Template of a given Type is then used in
11650 CREATing an Object of that Type. There is a single Object in the
11700 system whose Name and Type are both TYPE which represents all the
11750 Objects in the system (including itself) whose Type is TYPE. (See
11800 diagram)

11850

11900 The way of creating a new Object of some type, say FILE, is to use
11950 the Kall CREAT, supplying as an argument a FILE Template with CREARTS.
12000 A FILE Template can first be gotten by using the Kall TEMPLATE,
12050 supplying a Capability for the FILE TYPE Object with TEMPLTS.

12100

12150 Initially, HYDRA provides Templates for each Kernel Type (though
12200 users may not directly be able to access these). These Templates do
12250 not have all Kernel rights, but rather a restricted set, depending on
12300 the Type. For these rights limitations, see the Appendix.

12350

12400 CREAT may expect some additional arguments when creating an Object
12450 of a Kernel type. For instance, in CREATing a new TYPE Object, CREAT
12500 expects a Memory address as an additional argument. The Kernel will
12550 use the information in that block of memory to store the following
12600 data in the Data-Part of the TYPE Object:

12650

12700 * PNAME - the Type's Print Name. While all Objects have a 64 bit
12750 bit unique name, TYPE Objects also have a Print Name.

12800 The Kall WHAT, given a Capability, produces (among other
12850 information), the PNAME of its Type.

12900 * CAPINIT & CAPMAX - the initial length of the C-List (filled
12950 with Truenulls) and the maximum length of the C-List of
13000 any Object of the Type CREATed.

13050 * DATAINIT & DATAMAX - the initial length of the Data-Part (zeroed)
13100 and the maximum length of the Data-Part of any Object of
13150 the Type CREATed.

13200 * RTRVFLAG - An indication of whether Objects of this type are
13250 to be retrieved when all references to the Object are
13300 deleted (See following paragraph)

13350
13400
13450
13500
13550
13600
13650
13700
13750
13800
13850
13900
13950
14000
14050
14100
14150
14200
14250
14300
14350
14400
14450
14500
14550
14600
14650
14700
14750
14800
14850
14900
14950
15000
15050
15100
15150
15200
15250
15300
15350
15400
15450
15500
15550
15600
15650
15700
15750
15800
15850
15900
15950
16000

When all Capabilities for an Object have been deleted, the Object is normally garbage collected. However, it is possible to retrieve such Objects and prevent garbage collection on a Type by Type basis (see RTRVFLAG above). The Kall TYPRETRIEVE returns a Capability for an Object, all of whose references have been deleted (including aliases). To really garbage collect a retrievable Object, the Kall ERASE rather than DELETE must be used to delete the last Capability for the Object. Aliasing Objects are never retrieved.

.SUBSEC |PROTECTED SUBSYSTEMS|

Since Protected Subsystems are generally built around a particular type of Object (e.g. - the Directory Subsystem mentioned earlier), HYDRA provides a way to use a Subsystem without unnecessarily proliferating Capabilities for the Procedures which define it.

The C-List of a Type Object is used to implement protected subsystems easily by listing the Procedures which define it, and supplying access to those Procedures through the Kall TCALL.

If the Ndx'th Capability in the current LNS is of type T, and we use T[j] to denote the j'th Capability in the C-List of the T-Type Object, then TCALL(Rtrn,Ndx,j,a2,...,ak) is the same as CALL(Rtrn,T[j],Ndx,a2,...,ak). See the diagram.

.SUBSEC |SPECIFICATIONS FOR INTERMEDIATE KERNEL CALLS|

TEMPLATE MANIPULATION

TEMPLATE (Path, Nnew, Memr)

Parameters:

Path - Path index; Steps: LOADRTS,UCNFRTS;
Pretarget: STORTS,MDFYRTS; Target: Empty
Nnew - Simple index, Type TYPE, TEMPLRTS
- or a negative integer between -1 and -13
Memr - Legitimate Stack Memory address or 0

Effect: If Nnew is a Simple index, then TEMPLATE places a Template in Path's Target whose Type is the Name of the Nnew'th Capability in the Current LNS. The Template will have all flags and rights but FRZRTS & ALLYRTS.

If Nnew is negative, then a Template for the (-Nnew)'th Kernel Type is placed in Path's Target with TEMPLFLAG set as well as various rights depending on the Type. The first 13 types are the predefined Kernel Types.

In either case, the rights of the new Template are further restricted according to the contents of Memr (if Memr is nonzero).

Result: 0

16050
16100 SETCHKRTS (Path, Mem)
16150 Parameters:
16200 Path - Path index; Steps: LOADRTS,UCNFRTS;
16250 Pretarget: LOADRTS,STORTS,KILLRTS,MDFYRTS;
16300 Target: Template, DLTRTS
16350 Mem - Legitimate Stack Memory address
16400 Effect: Sets the Check-Rights of the Template at Index
16450 according to the contents of Mem.
16500 Result: 0
16550
16600
16650 OBJECT MANIPULATION
16700
16750
16800 CREAT (Nnew, Ncur, <arguments>)
16850 Parameters:
16900 Nnew - Simple index, Empty
16950 Ncur - Simple index, Template, CREATS; must not be NULL;
17000 Also requires UCNFRTS if the Type is Retrievable
17050 For description of additional arguments (only applicable
17100 when CREATing a Kernel Object) see the Appendix
17150 Effect: Creates a new Object of the same Type as Ncur and
17200 places a Capability for it in Nnew. The rights in
17250 Nnew are the same as those in Ncur plus DLTRTS.
17300 Result: 0
17350
17400
17450 COPY (Nnew, Ncur, <arguments>)
17500 Parameters:
17550 Nnew - Simple index, Empty
17600 Ncur - Simple index, Object Reference, COPYRTS
17650 For description of additional arguments (only applicable
17700 when COPYing a Kernel Object) see the Appendix
17750 Effect: Creates a new Object of the same type as Ncur
17800 and places a Capability for it in Nnew. In addition, the
17850 C-List and Data-Part of the new Object will be made the
17900 same as those of the original.
17950 The rights of the new Capability in Nnew will be exactly
18000 the same as those for Ncur plus DLTRTS, unless the Object
18050 is of a Kernel Type in which case additional rights may be
18100 added. See the Appendix for details.
18150 Result: 0
18200
18250
18300 SWITCH (Path, Ncur)
18350 Parameters:
18400 Path - Path index; Steps & Pretarget: LOADRTS,UCNFRTS;
18450 Target: Object Reference, MDFYRTS,OBJRTS
18500 Ncur - Simple index, same Type as Path's Target, OBJRTS,MDFYRTS
18550 or 0
18600 Effect: If Ncur is not zero, switches the C-List and Data-Part of
18650 the Objects referenced by Path's Target and Ncur. If Ncur
18700 is zero, destroys the Object referenced by the Target (same

18750 effect as ERASE).

18800 Future accesses of the Object will fail with either SCBND or

18850 SDBND signals.

18900 Signals:

18950 SLOCK - If the Object referenced by Ncur cannot be locked

19000 immediately

19050 Result: 0

19100

19150

19200 FREEZE(Ncur)

19250 Parameters:

19300 Ncur - Simple index, must be only extant reference to an

19350 Object, OBJRTS,UCNFRTS; Object must not be an Alias;

19400 Each Capability in C-List of Object must have FRZRTS

19450 Effect: Effectively freezes the Object by doing the following to

19500 the only Capability for the Object: Sets FRZRTS and

19550 turns off UCNFRTS & MDFYRTS.

19600 Signals:

19650 SFRZ - Some Capability in the Object's C-List is not frozen.

19700 SIGDATA indicates the index of the last such Capability.

19750 SUNQ - Ncur is not the only reference to the Object.

19800 SALIAS - Ncur references an Alias

19850 Result: 0

19900

19950

20000 ALIAS (Nnew, Ncur)

20050 Parameters:

20100 Nnew - Simple index, Empty

20150 Ncur - Simple index, Object Reference

20200 Effect: Creates an Object of the same type as Ncur to act as an

20250 Alias for the Object referenced by Ncur. Any future

20300 references to the new Object (unless changed by

20350 REALLY) will in fact access Ncur's Terminal Object. Nnew

20400 will have the same rights as Ncur except DLTRTS and ALLYRTS

20450 will be added and it will not have FRZRTS.

20500 Result: 0

20550

20600

20650 REALLY (Nnew, Ncur)

20700 Parameters:

20750 Nnew - Simple index, ALLYRTS (insures Aliasing Object)

20800 Ncur - Simple index, Object Reference of same type as Nnew,

20850 except for DLTRTS & ALLYRTS, must have at least all

20900 the rights as Nnew has.

20950 - or 0

21000 Effect: If Ncur is not zero, re-allies the Object referenced

21050 by Nnew to be an alias for the Object referenced by Ncur.

21100 If Ncur is zero, the Object referenced by Nnew will become

21150 an alias for nothing and future references to it will fail

21200 with signal SALLY.

21250 Result: 0

21300

21350

21400 TYPRETRIEVE (Nnew, Ncur)

21450 Parameters:
21500 Nnew - Simple index, Empty or 0
21550 Ncur - Simple index, TYPE Object Reference, UCNFRTS, RTRVRTS
21600 Effect: If Nnew is not zero, retrieves a Capability for an Object
21650 of Type Named by Ncur, all of whose references have been
21700 deleted. The Kernel maintains the retrieval queue for each
21750 Object in FIFO order. The retrieved Capability has all rights
21800 set except FRZRTS and ALLYRTS (Aliasing Objects are not
21850 retrieved). If Nnew is non-zero, the Kall is executed for
21900 its Result only.
21950 Result: Number of Objects in Ncur's Type's Retrieval queue
22000 (including Object retrieved - if any. Note a result of 0
22050 indicates no Object was retrieved).
22100
22150
22200 ERASE (Ncur)
22250 Parameters:
22300 Ncur - Simple index, must be only reference to Object, OBJRTS
22350 Effect: Deletes last reference to an Object without placing it in its
22400 Type's retrieval queue. Also deletes each Capability in the
22450 Object's C-List. (If the Capability is for an aliasing Object,
22500 or no retrieval is indicated for the type, simply
22550 deleting the last reference to the Object has the same
22600 effect as ERASEing it.)
22650 Signals:
22700 SUNQ - Ncur is not the only reference to the Object
22750 Result: 0
22800
22850
22900
22950 THE CALL MECHANISM
23000
23050
23100 MERGE (Nnew, Ntmpl, Path)
23150 Parameters:
23200 Nnew - Simple index, Empty
23250 Ntmpl - Simple index, Template, TEMPLFLAG
23300 Path - Path index; Pretarget: LOADRTS; Target: Defined,
23350 Rights must contain all those specified by Check-Rights
23400 field of Ntmpl. If Ntmpl is not Null, must be an
23450 Object Reference and must be of the same Type as Ntmpl.
23500 If Ntmpl is Null, may be of any Type and may be either
23550 an Object Reference or a Template.
23600 Effect: Copies the Capability targeted by Path to the Nnew'th slot
23650 of the current LNS and sets DLRTS. If Path's Target is a
23700 Capability for an Aliasing Object and Ntmpl has NEWFLAG set,
23750 a Capability for the Alias's Terminal Object is copied instead.
23800 If Ntmpl has NEWFLAG set, Ntmpl's rights are copied to
23850 Nnew, except for ENVRTS, UCNFRTS, MDFYRTS & FRZRTS which are
23900 the same as in Path's Target.
23950 If any Capability in the Path lacked UCNFRTS, then MDFYRTS,
24000 UCNFRTS & ALLYRTS will be removed from Nnew.
24050 Signals:
24100 SRTSM - Check-Rights failure

24150 SKNDT - Ntmpl is not a Template or does not have TEMPLFLAG set.
24200 STYPC - Types of Path's Target and Ntmpl are not the same.
24250 Result: 0
24300
24350
24400 MAKLNS (Nnew, Nproc, <arguments>)
24450 Parameters:
24500 Nnew - Simple index, Empty
24550 Nproc - Simple index, Procedure Object Reference
24600 - The 0 or more arguments must each be of the following form:
24650 1] Path - Path index; Pretarget: LOADRTS;
24700 Target: Requires ENVRTS if Nproc has PRCSRTS
24750 2] Restrict (Path, Memr) - Path is as for [1] and
24800 Memr is a Legitimate Stack Memory address or 0
24850 3] Transfer (Path, Memr) - Path is a Path index;
24900 Steps: UCNFRTS,LOADRTS;
24950 Pretarget: MDFYRTS,LOADRTS,KILLRTS;
25000 Target: DLTRTS, also requires ENVRTS if
25050 Nproc has PRCSRTS.
25100 Memr is a Legitimate Stack Memory address or 0
25150 4] Memdata (Memd, Knt) - Memd is a Legitimate
25200 Stack Memory address and Knt is a positive
25250 integer
25300 5] Stkdata (<data>) - <data> is 0 or more words
25350 of data
25400 The Capability denoted by each argument must also
25450 satisfy the requirements of its corresponding Parameter
25500 Template (see MERGE)
25550 Effect: An LNS is incarnated from the Procedure and arguments and
25600 a Capability for it is placed in Nnew with DLTRTS. In
25650 addition it will have UCNFRTS & FRZRTS, and the
25700 auxiliary rights LNSRTS & PRCSRTS if Nproc does.
25750 The LNS will be made Confined if Nproc lacks UCNFRTS. The
25800 LNS will be made Blind if Nproc has FRZRTS or if the Current
25850 LNS is Blind and Nproc lacks UCNFRTS.
25900 All Capabilities in the C-List of the PROCEDURE which are
25950 either Object References or Capability Templates (TEMPLFLAG set)
26000 are copied to the same slot in the C-List of the incarnated
26050 LNS. If Nproc lacks UCNFRTS, each of these will have UCNFRTS,
26100 MDFYRTS & ALLYRTS removed.
26150 Parameter Templates in the C-List of the PROCEDURE are
26200 Capabilities specified by the Arguments. Arguments are matched
26250 with Parameter Templates from last to first. If fewer arguments
26300 are specified than Parameter Templates, the additional Parameter
26350 slots at the beginning of the LNS may be filled by Nulls (See
26400 the Section of PROCEDURE & LNS CONTEXT BLOCKS for details).
26450 The Capabilities that will be placed in the parameter slots
26500 of the LNS are the result of MERGEing the Parameter
26550 Template with a Capability specified by the corresponding
26600 argument. For details of each individual merge, see the Effects
26650 part of the MERGE Kall. As noted, arguments come in 5 flavors.
26700 The Capabilities they specify and additional side effects are
26750 as follows:
26800 1] Capability is Path's Target

26850 2] Capability is Path's Target, restricted by Memr's contents
26900 if Memr is non-zero
26950 3] Capability is Path's Target, restricted by Memr's contents
27000 if Memr is non-zero. In addition, the Capability at Path's
27050 Target is deleted. (N.B. use wisely, since, even if the Kall
27100 fails, the Capability may be lost)
27150 4] Capability is for a newly created Data Object with all
27200 rights but FRZRTS & ALLYRTS. The Data-Part of the new
27250 Object will contain the Knt words of Data copied from the
27300 block of Memory beginning at Memd.
27350 5] Capability is for a newly created Data Object with all
27400 rights but FRZRTS & ALLYRTS. The Data-Part of the new Object
27450 will consist of '<data>'.
27500 Signals:

- 27550 - If an argument is bad or any merge failed, the usual signal
27600 will be generated with SLNS orred in as well. In addition,
27650 the fixed location SIGDATA in the stack page contains the
27700 index of the affected slot in the incarnated LNS in its low
27750 order byte and the number of the affected argument in its
27800 high order byte.
- 27850 SFARG - Too few arguments. SIGDATA indicates the minimum
27900 number of arguments acceptable.
- 27950 SMARG - Too many arguments. SIGDATA indicates the maximum
28000 number of arguments acceptable.
- 28050 SXCNF - LNS is not allowed to be made Confined.
28100 (See Section on PROCEDURE & LNS CONTEXT BLOCKS)
- 28150 SXBLND - LNS is not allowed to be made Blind

28200 Result: 0
28250
28300
28350 LNSCALL (Rtrn, Nlns)
28400 Parameters:

- 28450 Rtrn - Simple index, Empty
- 28500 Nlns - Simple index, LNS Object Reference, LNSRTS;
28550 The LNS must be "useable" (see Subsections on User
28600 Traps and Process Objects)

28650 Effect: The LNS is Called and execution begins in its
28700 environment. When the Called LNS KRETURNS, it may specify
28750 a Capability to be returned. If Rtrn is not zero, it
28800 designates the slot where that Capability will be put.
28850 If Rtrn is zero, a returned Capability is simply discarded.
28900 Signals:

- 28950 - For Paging related signals, see the Paging Section
- 29000 SSTK - Inadequate stack space available to run the LNS (See
29050 Section on PROCEDURE & LNS CONTEXT BLOCKS).
29100 SIGDATA contains amount of additional stack space needed.
- 29150 SCNTRL - Callee returned by 'Punting a Control' rather than
29200 a KRETURN (See PROCEDURE & LNS CONTEXT BLOCKS).
- 29250 SLOCK - LNS is currently in use (See PROCESS CREATION)
- 29300 SREUSE - LNS may not be Reused (See next Section)
- 29350 - When the Callee KRETURNS, it specifies a Return Value. If
29400 that value is negative, it is treated as a signal.

29450 Result: Value returned by the Callee
29500

29550
29600 CALL (Rtrn, Nproc, <arguments>)
29650 Parameters:
29700 Rtrn - Simple index, Empty or 0
29750 Nproc - Simple index, Procedure Object Reference, CALLRTS
29800 - Specifications for arguments are exactly
29850 as for MAKLNS. In addition to the 5 specified in MAKLNS,
29900 there are two more possible specifications:
29950 6] Lns
30000 7] Lnsrestrict (Memr) - Memr is a Legitimate
30050 Stack Memory address or 0
30100 Effect: The effect is almost equivalent to the sequence
30150 MAKLNS (*, Nproc, <arguments>); LNSCALL (Rtrn, *).
30200 That is, the Kernel incarnates the LNS and Calls it, without
30250 the Caller ever having a Capability itself for the incarnated
30300 LNS. The only difference is that, unless required by
30350 Check-Rights in a Paramter Template, an argument's target
30400 does not require ENVRTS, regardless of whether or not
30450 Nproc has PRCSRTS.
30500 The Capabilities denoted by the additional argument
30550 specifications noted above are:
30600 6] Capability is for the Caller's LNS with DLTRTS, MDFYRTS,
30650 UCNFRTS, LOADRTS, STORTS, APPRTS, KILLRTS, GETCBRTS, SETCBRTS,
30700 GSTKRTS and PSTKRTS.
30750 7] Capability is as in [6] with rights additionally
30800 restricted by the Memr's contents if Memr is non-zero.
30850 Signals:
30900 See MAKLNS & LNSCALL
30950 Result: Value returned by Callee
31000
31050
31100 KRETURN (Value, Ncur, Memr)
31150 Parameters:
31200 Value - Integer
31250 Ncur - Simple index, ENVRTS or 0
31300 Memr - Legitimate Stack Memory Address or 0
31350 Effect: Causes return of control to current LNS's Caller with
31400 result Value. If Value is negative, Value is signalled as
31450 well in the Caller's environment. If the Caller specified
31500 a Rtrn slot and Ncur is non-zero (and the return slot has
31550 not otherwise had a Capability STOREd into it), the
31600 Capability denoted by Ncur is returned to that slot in the
31650 Caller's LNS with rights restricted by the contents of Memr
31700 (if Memr is not zero) and with DLTRTS added.
31750 If the current LNS has no Caller, the current PROCESS will
31800 be stopped. Attempts to restart it will be unsuccessful.
31850 Result: Current value of R\$0. Control returns to Caller (unless a
31900 signal occurs). Control only continues normally after a
31950 KRETURN if the current LNS is subsequently LNSCALLED again.
32000
32050
32100
32150 PROTECTED SUBSYSTEMS
32200

32250
 32300 TLOAD (Nnew, Ncur, Ntyp)
 32350 Parameters:
 32400 Nnew - Simple index, Empty
 32450 Ncur - Simple index, Defined
 32500 Ntyp - Simple index into the C-List of the TYPE Object
 32550 whose Name is the Type of Ncur, Defined
 32600 - Current LNS must not be Blind
 32650 Effect: If Ncur is a Capability of Type T, then the Capability in
 32700 the Ntyp'th slot of the T TYPE Object is copied to the
 32750 Nnew'th slot of the current LNS with DLTRTS added. If
 32800 Ncur lacks UCNFRTS, then MDFYRTS, UCNFRTS & ALLYRTS will
 32850 be removed from Nnew.
 32900 Signals:
 32950 SBLND - Current LNS is Blind
 33000 Result: 0
 33050
 33100
 33150 TCALL (Rtrn, Ncur, Ntyp, <arguments>)
 33200 Parameters:
 33250 Rtrn - Simple index, Empty or 0
 33300 Ncur - Simple index, Defined
 33350 Ntyp - Simple index into the C-List of the TYPE Object
 33400 whose Name is the Type of Ncur, PROCEDURE Object
 33450 Reference, CALLRTS
 33500 - Current LNS must be Blind
 33550 Effect: The effect is exactly equivalent to the sequence
 33600 TLOAD (*, Ncur, Ntyp); CALL (Rtrn, *, <Ncur,<arguments>>).
 33650 That is, the Kernel CALLs the Procedure in the Type Object
 33700 without the Caller getting a Capability itself for the
 33750 Procedure. Ncur becomes the first argument of the CALL.
 33800 Signals:
 33850 See TLOAD & CALL
 33900 Result: Value returned by Callee
 33950
 ↑L

00050 .SEC |MORE ON PROCEDURES & LNS'S|

00100

00150 .SUBSEC |PROCEDURE & LNS CONTEXT BLOCKS|

00200

00250 The Data-Parts of PROCEDURES and LNS's are respectively known as
00300 Initial Context Blocks (ICB's) and Local Context Blocks (LCB's) and
00350 contain information relevant for execution and debugging. Information
00400 may not be gotten from or stored directly in Context Blocks using the
00450 standard Data-Part Kalls (GETDATA & PUTDATA), but rather specific
00500 Kalls (GETICB, SETICB, GETLCB & SETLCB) are used in conjunction with
00550 the auxiliary rights GETCBRTS and SETCBRTS. The list of fields in the
00600 Context Blocks, whether they can be read or written (in ICB or LCB),
00650 and their initial values (set at Procedure Creation time) can be found
00700 in the Appendix.

00750

00800 When an LNS is incarnated from a PROCEDURE, its LCB is copied from
00850 the ICB of the PROCEDURE, except for the field LVREG, which is set to
00900 the value of register R\$0 at incarnation time.

00950

01000 When one LNS Calls another, the general registers of the Caller are
01050 saved in its LCB, as well as the bounds of its active stack region and
01100 the contents of three fixed locations in the stack, SAVREG, SAVVAL and
01150 STKOWN, known collectively as SAVAREA. These values are all restored
01200 when the Called LNS returns. The SP, PS and PC are saved in fields LSP,
01250 LPS and LPC of the LCB. Registers 1-5 are saved in fields LR1 - LR5,
01300 Register 0 is saved in LVREG, the upper bound of the active stack is
01350 saved in SPUFLO and the three fields of the SAVAREA are saved in SVREG,
01400 SVVAL and SVOWN.

01450

01500 When the Callee begins execution, its PC, PS and R\$0 are initialized
01550 from the LCB (Paging information which determines the LNS's Page Set
01600 is also taken from the LCB - See the PAGING SECTION for more Details).
01650 When the Callee KRETURNS, R\$0 and the PC are saved in the LCB (as well
01700 as Paging information), thus if the LNS is LNSCALLED again, execution
01750 will continue immediately following the KRETURN, though except for R\$0
01800 and the PC, the other registers will be clobbered and the stack and
01850 Page Set will be reinitialized.

01900

01950

02000 .SUBSEC |USER TRAPS|

02050

02100 The LCB contains a number of user trap addresses which indicate the
02150 PC at which execution should continue after a Trap. Some of the traps
02200 roughly parallel the PDP-11 hardware (such as EMT & IOT) while
02250 others are provided by the HYDRA 'Virtual Machine'. Whenever a Trap
02300 is taken, the current PS and PC are pushed on the stack and execution
02350 proceeds at the Trap PC address with the PS same as the current PS
02400 except that Trace Trap Enable (bit 4) is turned off if it was on. The
02450 PS has the following format:

02500

02550

Bit	Meaning
-----	---------

02600

02650

0-3	Condition Codes
-----	-----------------

02700

4	Trace Trap Enable
---	-------------------

02750	5-7	Hardware Priority
02800	8-9	Hardware Space
02850	12	Reuse Flag
02900	13	Confined Flag (0 if Confined)
02950	14	Blind Flag (0 if Blind)
03000	15	Error Flag

03050

03100

The PDP-11 RTI instruction may be used to restore the old PC and PS. Bits 0-3, 4, 12 and 15 may have been changed in the stacked PS in any way. However, the Kernel checks RTI's and guarantees that fields 5-7, 8-9, 13 and 14 do not have values greater than when the LNS was incarnated.

03150

03200

03250

03300

03350

The following Trap PC fields are used for Hardware traps:

03400

03450

EMTPC - EMT instruction

03500

BKTPC - BKT instruction

03550

TRCPC - Trace Trap

03600

IOTPC - IOT instruction

03650

03700

In addition,

03750

03800

SIGPC - Signal PC, used when a Kall produced a signal

03850

03900

For all of the above Kalls, if the Trap PCs are 0 (especially important for signals), no Trap is performed.

03950

04000

04050

Any hardware error that occurs while the user is executing causes a Trap to the PC found in ERRPC. In addition, after the trap is taken, the Error Flag is turned on in the current PS. It can be cleared by RTI'ing with a PS in which Error Flag is not set (such as the one pushed on the stack when the error trap was taken). An error that occurs while the Error Flag is set (instead of causing a new trap) causes the process to be stopped. If ERRPC is zero, the trap is not dismissed; again, the process is stopped. In any case, the reason for the error is or'ed into the fixed location ERRCODE in the stack (See the Appendix for the meanings of the various error codes).

04100

04150

04200

04250

04300

04350

04400

04450

04500

04550

04600

The PRMASK is a mask of Processors on which the LNS can run. The mask is necessary since all C.mmp processors are not identical. Some have hardware floating point arithmetic, some run faster than others, and some may have a writable control store. If none of the needed processors are up, an Error will be caused. The PRMASK will be set to all 1's, and the old PRMASK will be put in SIGDATA.

04650

04700

04750

04800

04850

04900

04950

The CONTROL Kall (See next section) provides an inter-process interrupt mechanism. It is meant to be used only for debugging and 'emergency' situations. The Kernel Objects PORTs and POLSEMs are meant to be used by users for interprocess communication and signalling. The CTLMASK field in the LCB is a mask of those control interrupts the current LNS will accept (there are 16 bits, hence, 16 different control interrupts). Regardless of the contents of CTLMASK, a Blind LNS will accept no interrupts. Any interrupt not accepted simply pends till it is accepted. CTLPC contains the Control Trap address. The control

05000

05050

05100

05150

05200

05250

05300

05350

05400

05450 interrupts accepted will be or'ed into the fixed location CTLCODE in the
05500 stack . If CTLPC contains a 0, the current LNS will be forced to return,
05550 giving an SCNTRL signal to the Caller. In addition, all Controls
05600 indicated in CTLCODE will be re-controlled and thus may affect the
05650 Caller (as well as any control interrupts pending). This is known as
05700 'Punting a Control'.
05750

05800 Control interrupts may also be used as part of a more desperate
05850 debugger. Before CTLPC is checked, the contents of CTLCODE are
05900 compared against the field DBGMASK. If any bits match, a debugging
05950 PROCEDURE is Called that will have complete access to the environment
06000 of the current LNS.
06050

06100 If DBGMASK matches any bits of CTLCODE, the contents of the field
06150 DBGNDX in the LCB is used to index the current LNS's C-List. It
06200 should denote a Capability for a Procedure Object with CALLRTS. If so,
06250 the Procedure is CALLED with one argument, a Capability for the
06300 current LNS (see the LNS specification in CALL). If the CALL results
06350 in any kind of Signal, the CTLPC trap is taken, otherwise, CTLPC is
06400 completely ignored.
06450

06500 Since the Debugging Procedure is incarnated with an argument for the
06550 LNS to be debugged, it can manipulate and access its C-List, its LCB
06600 (via SETLCB & GETLCB) and its stack (via the Kalls GETSTACK &
06650 PUTSTACK) - in short anything the executing LNS could do itself.
06700

06750 After execution of the Debugging Procedure, the value of R\$0 will be
06800 restored from LVREG of the current LCB just as are the other registers.
06850 Thus, unless LVREG is changed by a SETLCB executed by the Debugging
06900 Procedure, R\$0 will be the same as it was before the Control Interrupt
06950 was accepted. The value returned by the Debugging Procedure is only
07000 inspected to determine if it is negative, in which case, as a signal
07050 return, it forces execution to continue at CTLPC as noted above.
07100

07150 It should be noted that Capabilities for LNS's with access rights
07200 are only generated in CALLs, and thus it is impossible to access any
07250 LNS (except the current executing one) while that LNS is executing.
07300

07350

.SUBSEC |THE PS AND THE STACK|

07400

07450 The subsection on User Traps noted how RTI's were restricted in some
07500 ways so that the current PS would not become more priveleged than when
07550 the current LNS was called. The PS of another LNS (given a Capability
07600 for that LNS with SETCBRTS) can be modified as well, through modifying
07650 the field LPS with the Kall SETLCB. The restriction on fields 5-7,
07700 8-9, 13 and 14 are the same.
07750

07800

07850 Bit 12 of the LPS field is the Reuse Flag. It controls whether a
07900 KRETURNed LNS can be reused, either through a subsequent LNSCALL or by
07950 using the LNS to initialize a Process. Only if bit 12 is set may it
08000 be reused.
08050

08100

The LPS field of an ICB can be set as well. The restriction is that

08150 the priority and space fields (5-7 & 8-9) can be set no greater than
08200 those of the current PS. Bits 13 & 14 of the LPS in the ICB act as
08250 incarnation and Call requirements. If bit 13 is set, then Confined
08300 incarnations of the ICB's PROCEDURE are not allowed. If bit 14 is
08350 set, then Blind incarnations of the ICB's PROCEDURE are not allowed.
08400
08450 All LNS's in a Process use the same Stack Page. However, the stack
08500 is protected so that one LNS cannot access another's stack except
08550 through the Calls GETSTACK and PUTSTACK. When an LNS Calls another
08600 LNS, the current bounds of its stack are stored in the LCB. SPUFLO
08650 (which cannot be altered) contains its upper bound, and LSP contains
08700 its lower bound. LSP can be changed as long as it is set below SPUFLO
08750 and above the address KALBND (See the appendix for the actual address
08800 of KALBND).
08850
08900 The active stack of an LNS which is not executing extends from
08950 SPUFLO to the value of SP when the LNS Called its Callee - #20.
09000 PUTSTACK can (given a Capability for an LNS) modify any portion of its
09050 active stack. The additional #20 bytes at the bottom of the stack
09100 provide a small area in which a debugger can extend the stack. Note
09150 that the actual value of LSP can be set even below that, but data
09200 cannot be put there. This is because it would run into the top of the
09250 stack of the LNS's Callee.
09300
09350 The field STKGROW is an estimate of the stack needed by an executing
09400 LNS. If not enough space is available on the stack to permit that
09450 much growth of the stack, the signal SSTK will be given when an
09500 attempt is made to Call the LNS.
09550
09600
09650 .SUBSEC |MORE ON CONTEXT BLOCKS|
09700
09750 There is often a need to allow PROCEDUREs to accept a variable
09800 number of arguments when Called. If fewer arguments are passed to a
09850 Procedure than there are Parameter Templates, then, if the number of
09900 arguments is greater than or equal to the value of field ARGMIN in the
09950 ICB, the Call will succeed and the unfilled Parameter Templates will
10000 be filled with Nulls in the LNS; otherwise, the Call fails with signal
10050 SFARG.
10100
10150 ARGCALL in the LCB contains the actual number of arguments used in
10200 incarnating the LNS.
10250
10300 RTRNDX contains the index in the LNS that Called this one where a
10350 returned Capability will be placed.
10400
10450 PROCDATA is an 8 word field that can be used to identify the
10500 PROCEDURE. It is modifiable in the ICB, but when copied into the
10550 corresponding field of an incarnated LNS, it is not modifiable. The 8
10600 word field LNSDATA is writable in both.
10650
10700 The remainder of the fields in the ICB/LCB have to do with Paging
10750 and are described in the Paging Section.
10800

10850
10900 .SUBSEC |SPECIFICATION FOR CONTEXT BLOCK KALLS|
10950
11000
11050 GETICB (Memd, Path, Code)
11100 Parameters:
11150 Memd - Legitimate Stack Memory address
11200 Path - Path index; Pretarget: LOADRTS;
11250 Target: PROCEDURE Object Reference, GETCBRTS
11300 Code - Positive integer, legitimate code
11350 Effect: Copies information from the Initial Context Block of
11400 the Procedure into a block of Memory beginning at Memd.
11450 The content and amount of information copied depends
11500 on the Code. For legitimate codes and what gets copied,
11550 see the Appendix.
11600 Signals:
11650 SCODE - Bad Code
11700 Result: 0
11750
11800
11850 SETICB (Path, Memd, Code)
11900 Parameters:
11950 Path - Path index; Steps & Pretarget: LOADRTS,UCNFRTS;
12000 Target: PROCEDURE Object Reference, SETCBRTS,MDFYRTS
12050 must reference a PROCEDURE Object.
12100 Memd - Legitimate Stack Memory address
12150 Code - Positive integer, legitimate code
12200 Effect: Uses information in the block of Memory beginning at
12250 Memd to set various values in the Initial Context
12300 Block. For legitimate codes and their effects, see the
12350 Appendix.
12400 Signals:
12450 SCODE - Bad Code
12500 SLPS - Bad PS (See Subsection on PS & the Stack)
12550 Result: 0
12600
12650
12700 GETLCB (Memd, Path, Code)
12750 Parameters:
12800 Memd - Legitimate Stack Memory address
12850 Path - Path index; Pretarget: LOADRTS;
12900 Target: LNS Object Reference, GETCBRTS
12950 - or 0
13000 Code - Positive integer, legitimate code
13050 Effect: Copies information from the Local Context Block of
13100 the LNS into a block of Memory beginning at Memd (If Path
13150 is 0, then the current executing LNS is used).
13200 The content and amount of information copied depends
13250 on the Code. For legitimate codes and what gets copied,
13300 see the Appendix.
13350 Signals:
13400 SCODE - Bad Code
13450 Result: 0
13500

13550
 13600 SETLCB (Path, Memd, Code)
 13650 Parameters:
 13700 Path - Path index; Steps & Pretarget: LOADRTS,UCNFRTS
 13750 Target: LNS Object Reference, SETCBRTS,MDFYRTS
 13800 - or 0
 13850 Memd - Legitimate Stack Memory address
 13900 Code - Positive integer, legitimate code
 13950 Effect: Uses information in the block of Memory beginning at
 14000 Memd to set various values in the Local Context
 14050 Block of the LNS (if the Path is 0, then the current
 14100 executing LNS is used). For legitimate codes and their effects,
 14150 see the Appendix.
 14200 Signals:
 14250 SCODE - Bad Code
 14300 SLPS - Bad PS
 14350 SLSP - Bad SP
 14400 Result: 0
 14450
 14500
 14550 GETSTACK (Memd, Ilns, Meml, Knt)
 14600 Parameters:
 14650 Memd - Legitimate Stack Memory address
 14700 Ilns - Simple index, LNS Object Reference, GSTKRTS
 14750 Meml - Legitimate Stack Memory address in the active
 14800 stack of the LNS denoted by Ilns.
 14850 Knt - Positive integer
 14900 Effect: Moves up to Knt words of data from Meml to Memd. Fewer
 14950 than Knt words will be copied if there are fewer than
 15000 Knt words above and including Meml in Ilns's active stack.
 15050 Signals:
 15100 SLMEM - Meml is a bad stack address
 15150 Result: Number of words copied
 15200
 15250
 15300 PUTSTACK (Ilns, Meml, Memd, Knt)
 15350 Parameters:
 15400 Ilns - Simple index, LNS Object Reference, PSTKRTS,MDFYRTS
 15450 Meml - Legitimate Stack Memory address in the active stack
 15500 of the LNS denoted by Ilns
 15550 Memd - Legitimate Stack Memory address
 15600 Knt - Positive integer
 15650 Effect: Moves Knt words of data from Memd to Meml.
 15700 Signals:
 15750 SLMEM - Meml is a bad stack address
 15800 Result: 0
 15850
 ↑L

00050 .SEC |PROCESSES, POLICIES & SEMAPHORES|

00100

00150 .SUBSEC |PROCESS OBJECTS|

00200

00250 Process Objects are the scheduling entities of the HYDRA Kernel.
00300 Unlike many systems, there is no explicit process hierarchy in HYDRA.
00350 To stop or start a process, one merely needs a Capability for the
00400 Process with the appropriate rights. Starting or stopping of one
00450 process has no effect on any other process.

00500

00550 Process creation is accomplished using the Kall CREAT already
00600 described.

00650

00700 CREAT (Nnew, Nprcs, Nlns) - Creation of Process Object

00750 Parameters:

00800

Nnew - Simple index, Empty

00850

Nprcs - Simple index, PROCESS Template, CREARTS

00900

Nlns - Simple index, LNS Object Reference, PRCSRTS;

00950

The LNS must be "useable" (not currently active in an

01000

LNSCALL or Process CREAT which has not yet returned,

01050

and must have its REUSE Flag set if it has already been

01100

LNSCALLED and subsequently returned).

01150

Effect: Creates a PROCESS Object and places a Capability for

01200

it in Nnew. The rights in Nnew are the same as those

01250

in Nprcs plus DLRTS.

01300

The LNS referenced by Nlns provides the initial environment

01350

(LNS) of the Process when it is first STARTed.

01400

Signals:

01450

- For Paging related signals, see the Paging Section

01500

SLOCK - LNS currently active

01550

SREUSE - LNS may not be reused

01600

Result: 0

01650

01700

01750 .SUBSEC |THE PROCESS BASE|

01800

01850 Optionally associated with a Process is a Process Base, a UNIVERSAL
01900 Object that remains associated with the Process over calls and
01950 returns. The Kall BLOAD loads a Capability from the current Process's
02000 Base into the current LNS and BCALL CALLs a Procedure in the Process
02050 Base. A Process Base can be used to provide generally available
02100 facilities to a Process or more likely, a group of processes.

02150

02200

If an LNS is confined, the Capabilities in the Process Base act as
02250 though they lacked UCNFRTS. If an LNS is Blind, the Process Base may
02300 not be used.

02350

02400

02450 .SUBSEC |POLICY SUBSYSTEMS & LONG-TERM SCHEDULING|

02500

02550 Before a Process is able to run, it must be associated with a POLICY
02600 Object via the POLICY Kall (which also can associate a Process with
02650 its Base). Processes have specific resource needs, space (both for
02700 pages, in core and out, and for Objects) and cpu time. POLICY Objects

02750 provide the mechanism for allocation of these resources. By a 'Policy
02800 Subsystem', we mean the set of Procedures that manage the scheduling
02850 and allocation of the Processes associated with a particular Policy
02900 Object.

02950
03000 To allow multiple Policy Subsystems to coexist, each Policy Object
03050 is provided (via the Kall MAKEPOLICY) with resource guarantees (a
03100 percentage of CPU-time and memory allocation guarantees). In turn, a
03150 Policy Subsystem may fix memory guarantees for each process associated
03200 with it, which acts as an upper limit to the memory resources the
03250 process may use when running.

03300
03350 The Kalls START and STOP start and stop Processes and are the means
03400 by which a Policy Subsystem implements long-term scheduling.

03450
03500 The Kall START (given a Capability for a PROCESS with STARTS) swaps
03550 a process's pages into memory and makes the process available for
03600 execution. STARTing a Process associated with a POLICY Object P will
03650 fail, if the Process's memory guarantee added to the sum of the
03700 Process memory guarantees of all the running Processes associated with
03750 P exceeds P's memory guarantee.

03800
03850 When a Process is stopped, either by the Kall STOP or for some other
03900 reason, its pages may be swapped out and the memory allocated to it is
03950 made available for reallocation by the Policy Subsystem.

04000
04050

04100 .SUBSEC |KMPS & THE PCB|

04150

04200 After a Process is started and until it is stopped, short-term
04250 scheduling is provided by KMPS, the Kernel MultiProgramming System. A
04300 Policy Subsystem can affect KMPS's scheduling by setting some fields
04350 (FPRIORITY, FNSLICES & FSLICE) in the Data-Part of the Process, its
04400 PCB (Process Context Block).

04450

04500 The fields in the PCB which affect KMPS scheduling are:

04550

04600 PRMASK - Processor mask, a mask of the processors upon which the
04650 Process may run. It is the same as the PRMASK of the LNS currently
04700 executing under the Process.

04750

04800 PRIORITY - Relative importance of a Process. When a processor
04850 becomes available, KMPS first chooses a Policy Object and then runs
04900 the highest priority Process associated with that Policy that can run
04950 on the processor. If the high order bit of PRIORITY is 1, the Process
05000 will not be stopped when it runs out of time (i.e. NSLICES & SLICE are
05050 ignored).

05100

05150 NSLICES, SLICE - Number of time slices & time slice size (in
05200 microseconds). KMPS will run a Process for NSLICES time slices of
05250 SLICE size each. When the process has used up its total time quantum,
05300 it is stopped, and must be reSTARTed before KMPS will schedule it
05350 again.

05400

05450 In addition, KMPS contains the following fields:
05500
05550 POLID - A word used by a Policy Subsystem to identify the Process
05600 (see THE POLICY QUEUE).
05650
05700 CPSPMAX - Core Page guarantee. Maximum number of pages in the
05750 working set of any LNS executing under the Process.
05800
05850 CPSCUR - Number of pages in current working set.
05900
05950 TIMER - Remaining time in current slice.
06000
06050 NUSLICES - Number of time slices used (cleared when the Process is
06100 STARTed).
06150
06200 RSTATE - Running state. There are four possibilities:
06250 0 - RUNNING. Process is actually running on a Processor.
06300 1 - FEASIBLE. Process is in KMPS waiting to run.
06350 2 - BLOCKED. Process is in KMPS but blocked.
06400 3 - STOPPED. Process is not in KMPS.
06450
06500 RCVCODE - Policy Receive Code (See THE POLICY QUEUE). Contains bits
06550 indicating additional status of the process, including reasons why the
06600 process has been stopped. More than one bit may be set (See Appendix
06650 for meanings of each bit). The field is cleared when the process is
06700 restarted.
06750
06800 CTLMASK - Controls accepted by the LNS executing under the Process.
06850
06900 CTLCODE - Controls pending. A Control interrupt may be sent to a
06950 stopped process. If it matches any bits in CTLMASK, it will strike as
07000 soon as the Process begins running. Any control interrupts not
07050 accepted by CTLMASK will continue to pend until accepted by a change
07100 of CTLMASK.
07150
07200
07250 .SUBSEC |EXECUTION PROTECTION|
07300
07350 Though HYDRA/C.mmp has been designed to be an extremely reliable
07400 system, a hardware failure can halt the execution of an LNS at an
07450 arbitrary time. Hence, users should adopt (in general) the MULTICS
07500 philosophy: When operating on sensitive information, leave enough
07550 audit information around so that a recovery procedure can complete the
07600 operation regardless of where in the operation a crash might have
07650 occurred.
07700
07750 More generally, while a user may build his own Policy Subsystem, it
07800 is likely that he will elect to use one made generally available to
07850 the user community. A Process may be STOPped at any time, and it is
07900 certainly within the range of possibility (especially using a buggy
07950 Policy Subsystem) that the Process may never be restarted.
08000
08050 A Policy Subsystem also has available the CONTROL Kall to send
08100 interrupts to a Process. A buggy subsystem may send so many

08150 interrupts that the executing LNS will spend all of its time fielding
08200 the control interrupts.

08250

08300 To solve all of these problems (except for the problem of unexpected
08350 crashes), the RUNTIME Kall is provided. RUNTIME specifies an amount
08400 of time during which the current Process will neither be stopped nor
08450 will receive any Control interrupts. RUNTIME also solves a more
08500 useful problem, to wit: Consider a Data Base that is accessed and
08550 changed frequently by cooperating concurrent processes. If access and
08600 modification are fast operations, then if the operations are
08650 execution-protected by RUNTIME, a busy-wait lock which is part of the
08700 Data Base may suffice to provide mutual exclusion rather than more
08750 complex (though better structured) use of synchronization objects
08800 (SEMAPHOREs, POLSEMs & PORTS).

08850

08900 Some uncertainties about execution can be resolved if a user has
08950 some information about the Policy Subsystem and its status with
09000 which her program executes. The Kall INFPOLICY returns
09050 a word that reflects such information. The value of that word is set
09100 when the POLICY Object was created.

09150

09200

09250 .SUBSEC |SEMAPHORES|

09300

09350 SEMAPHORE Objects are supplied to provide short term synchronization
09400 for trusted Subsystems. In general, users will not have Capabilities
09450 for Semaphores but will use POLSEMs (POLicy SEMaphores) and PORTs
09500 instead.

09550

09600 Semaphore Objects are created with an initial count (parameter for
09650 Semaphore CREAT) that specifies the number of PSEM's more than VSEM's
09700 that may be executed without causing the Process to wait. A Process
09750 waiting on a SEMAPHORE is not stopped, and in fact, cannot be STOPped
09800 (and thus swapped out) until it passes the SEMAPHORE.

09850

09900 When a SEMAPHORE is erased, it is first V'd as many times as are
09950 necessary to wake up all Processes waiting on the Semaphore.

10000

10050 For reliability, a limit is set for the amount of time a Process may
10100 be blocked on a SEMAPHORE. If the Process is blocked for a longer
10150 time, the Process continues execution and its PSEM (the Kall which P's
10200 a Semaphore) fails.

10250

10300

10350 .SUBSEC |THE POLICY QUEUE|

10400

10450 The Kernel keeps a queue for each POLICY Object. When a Process
10500 stops, information about the stopped process is placed in the POLICY
10550 queue. The Kall RCVPOLICY is used to extract an entry from the Policy
10600 queue in FIFO order. (The Policy queue is also used for other Process
10650 related messages. See the section on PORTS & POLSEMS for further
10700 details). The information extracted includes POLID so that the Policy
10750 Subsystem can identify the Process affected.

10800

10850
10900 .SUBSEC |SPECIFICATIONS FOR PROCESS, SEMAPHORE & POLICY CALLS|
10950
11000
11050 PROCESS CONTEXT BLOCKS
11100
11150
11200 GETID
11250 Parameters:
11300 - Current LNS must not be Blind
11350 Effect: None
11400 Signals:
11450 SBLND - Current LNS is Blind
11500 Result: Process ID of the current Process
11550
11600
11650 GETPCB (Memd, Path, Code)
11700 Parameters:
11750 Memd - Legitimate Stack Memory address
11800 Path - Path index; Pretarget: LOADRTS;
11850 Target: PROCESS Object Reference, GETCBRTS
11900 Code - Positive integer, legitimate code
11950 Effect: Copies information from the Process Context Block of
12000 the Process into a block of Memory beginning at Memd.
12050 The content and amount of information copied depends
12100 on the Code. For legitimate codes and what gets copied,
12150 see the Appendix.
12200 Signals:
12250 SCODE - Bad Code
12300 Result: 0
12350
12400
12450 SETPCB (Path, Memd, Code)
12500 Parameters:
12550 Path - Path index; Steps & Pretarget: LOADRTS,UCNFRTS
12600 Target: PROCESS Object Reference, SETCBRTS,MDFYRTS;
12650 Unless the PROCESS is the current one, the PROCESS
12700 must be stopped.
12750 Memd - Legitimate Stack Memory address
12800 Code - Positive integer, legitimate code
12850 Effect: Uses information in the block of Memory beginning at
12900 Memd to set various values in the Process Context
12950 Block. For legitimate codes and their effects, see the
13000 Appendix.
13050 If current PCB is being changed, then any current RUNTIME
13100 is cancelled.
13150 Signals:
13200 SPRCS - Process not stopped
13250 SCODE - Bad Code
13300 Result: 0
13350
13400
13450
13500 PROCESS BASE

13550
13600 BLOAD (Nnew, Ncur)
13650 Parameters:
13700 Nnew - Simple index, Empty
13750 Ncur - Simple index into the current Process's Base, Defined
13800 - Current LNS must not be Blind
13850 Effect: Copies the Ncur'th Capability from the current Process Base
13900 to the Nnew'th slot of the current LNS adding DLTRTS. If the
13950 current LNS is Confined, Nnew will lack UCNFRTS.
14000 Signals:
14050 SKNDC - No Process Base
14100 SBLND - Current LNS is Blind
14150 Result: 0
14200
14250
14300 BCALL (Rtrn, Ncur, <arguments>)
14350 Parameters:
14400 Rtrn - Simple index, Empty or 0
14450 Ncur - Simple index into the current Process's Base,
14500 PROCEDURE Object Reference, CALLRTS
14550 - Current LNS must not be Blind
14600 Effect: The effect is exactly equivalent to the sequence
14650 BLOAD (*, Ncur); CALL (Rtrn, *, <arguments>).
14700 That is, the Kernel CALLs the Procedure in the Process
14750 Base without the Caller getting a Capability itself for
14800 the Procedure.
14850 Signals:
14900 See BLOAD & CALL
14950 Result: Value returned by Callee
15000
15050
15100
15150 SCHEDULING & CONTROL
15200
15250
15300 START (Nprcs)
15350 Parameters:
15400 Nprcs - Simple index, PROCESS Object Reference, STARTS,UCNFRTS;
15450 Process must be stopped but runnable
15500 Effect: Pages in the Process and enters it in KMPS
15550 Signals:
15600 SPRCS - Process is not Stopped
15650 SPOL - Process not associated with Policy Object
15700 SPOP - Initial LNS of Process has returned
15750 SGUAR - Policy Object guarantee has been exceeded. SIGDATA
15800 contains more information (See Appendix).
15850 Result: 0
15900
15950
16000 STOP (Nprcs, Code)
16050 Parameters:
16100 Nprcs - Simple index, PROCESS Object Reference, STOPRTS,UCNFRTS;
16150 Process must be in KMPS
16200 Code - Integer

16250 Effect: Removes Process from KMPS and enters an entry (including
16300 Code - called the Rcvcode) in the associated Policy's
16350 RCVPOLICY queue.
16400 Result: 0
16450
16500
16550 CONTROL (Nprcs,, Code)
16600 Parameters:
16650 Nprcs - Simple index, PROCESS Object Reference, CTLRTS,UCNFRTS
16700 - or 0
16750 Code - Integer
16800 Effect: Causes Control interrupts specified by Code to be sent to
16850 the Process (Current process if Nprcs is 0). See Subsection
16900 on User Traps.
16950 Result: 0
17000
17050
17100 RUNTIME (Tim)
17150 Parameters:
17200 Tim - Integer
17250 - Current LNS must not be Blind
17300 Effect: If Tim is zero, forces KMPS to reconsider its scheduling,
17350 which will cause a runnable process at the same or higher
17400 priority to run instead. In addition, though CTLMASK & PRMASK
17450 may be changed in the current LCB, the change only becomes
17500 effective if a RUNTIME (or call or return) is executed.
17550 RUNTIME also provides for uninterrupted execution. During
17600 that time the process may not be stopped (except due to errors,
17650 WORKSET and PPOLSEMs) and no Control interrupts are accepted.
17700 If Tim is positive, then if Tim is available in the total
17750 time remaining in the current and all remaining time slices,
17800 then execution proceeds uninterruptably (except for
17850 short term rescheduling by KMPS). Tim is in 1/2 seconds up
17900 to 1 minutes.
17950 If Tim is negative, then if -(Tim) is available in the
18000 current time slice, execution proceeds uninterruptably
18050 (except for hardware device interrupt handling). If -(Tim)
18100 is not available in the current time slice, but is less than
18150 or equal to the time slice size and at least one time slice
18200 remains, then before uninterrupted execution begins, the current
18250 time slice is ended and rescheduling is considered (but the
18300 process may not be STOPped or Control Interrupted). -(Tim)
18350 is in 16 microseconds up to 1/2 second.
18400 In either case, if the requested time is not available,
18450 the process is stopped. When reSTARTed, if the PCB has not
18500 been changed to make the requested time available, the Kall
18550 fails.
18600 If RUNTIME succeeds and a subsequent RUNTIME is executed
18650 in the uninterruptable period, pending STOP's and Control
18700 interrupts are re-enabled before the new RUNTIME takes effect.
18750 Signals:
18800 STIM - Requested time not made available
18850 SBLND - Current LNS is Blind
18900 Result: 0

18950
19000
19050
19100 SEMAPHORES
19150
19200
19250 PSEM (Path , Tim)
19300 Parameters:
19350 Path - Path index; Steps & Pretarget: LOADRTS,UCNFRTS;
19400 Target: SEMAPHORE Object Reference, MDFYRTS
19450 Tim - Positive integer
19500 Effect: P's the Semaphore
19550 Signals:
19600 SSEM - Process has been blocked on the Semaphore for more than
19650 Tim seconds.
19700 Result: 0
19750
19800
19850 CPSEM (Path)
19900 Parameters:
19950 Path - Path index; Steps & Pretarget: LOADRTS,UCNFRTS;
20000 Target: SEMPAHORE Object Reference, MDFYRTS
20050 Effect: Conditionally P's the Semaphore. The P is only executed if
20100 the process will not have to wait on it.
20150 Result: 1 if the P was executed, 0 if not.
20200
20250
20300 VSEM (Path)
20350 Parameters:
20400 Path - Path index; Steps & Pretarget: LOADRTS,UCNFRTS;
20450 Target: SEMAPHORE Object Reference, MDFYRTS
20500 Effect: V's the Semaphore
20550 Result: 0
20600
20650
20700 VASEM (Path)
20750 Parameters:
20800 Path - Path index; Steps & Pretarget: LOADRTS,UCNFRTS;
20850 Target: SEMAPHORE Object Reference, MDFYRTS
20900 Effect: V's the Semaphore exactly as many times as are needed to
20950 wake up all Processes waiting on it.
21000 Result: Number of V's done
21050
21100
21150
21200 POLICY KALLS
21250
21300
21350 POLICY (Nprcs, Npol, Nuniv)
21400 Parameters:
21450 Nprcs - Simple index, PROCESS Object Reference, MDFYRTS;
21500 If Npol is non-zero, requires POLRTS;
21550 If Nuniv is non-zero, requires BASERTS
21600 Npol - Simple index, POLICY Object Reference, POLRTS,MDFYRTS

21650 - or 0
 21700 Nuniv - Simple index, UNIVERSAL Object Reference, ENVRTS
 21750 - or 0
 21800 Effect: If Npol is non-zero, associates POLICY with the PROCESS.
 21850 If Nuniv is non-zero, makes the UNIVERSAL Object the
 21900 Process's Base.
 21950 Result: 0
 22000
 22050
 22100 RCVPOLICY (Memd, Npol)
 22150 Parameters:
 22200 Memd - Legitimate Stack Memory address
 22250 Npol - Simple index, POLICY Object Reference, RCVRTS, MDFYRTS
 22300 Effect: Extracts an entry from the Policy's queue and puts the
 22350 information from the entry into the 16 word area in memory
 22400 beginning at Memd.
 22450 If the queue is empty, the Process waits until an entry
 22500 arrives.
 22550 Result: 0
 22600
 22650
 22700 MAKEPOLICY (Nnew, Ncur, Memd)
 22750 Parameters:
 22800 Nnew - Simple index, POLICY Object Reference, MAKERTS, MDFYRTS
 22850 Ncur - Simple index, POLICY Object Reference, MAKERTS, MDFYRTS;
 22900 Memd - Legitimate Stack Memory address
 22950 Effect: Transfers allocations and guarantees between the two
 23000 POLICY Objects. The 16 word block beginning
 23050 at Memd contains information about how allocations and
 23100 guarantees are to be transferred.
 23150 Signals:
 23200 SGUAR - Bad guarantee specification. SIGDATA indicates
 23250 what was wrong. See Appendix for Details.
 23300 Result: 0
 23350
 23400
 23450 WHATPOLICY (Memd, Npol)
 23500 Parameters:
 23550 Memd - Legitimate Stack Memory address
 23600 Npol - Simple index, POLICY Object Reference
 23650 Effect: Information about the guarantees and allocations of the
 23700 POLICY Object is put into the 16 word area beginning at Memd.
 23750 Result: 0
 23800
 23850
 23900 INFPOLICY ()
 23950 Parameters:
 24000 - Current LNS must not be Blind
 24050 Effect: None
 24100 Signals:
 24150 SBLND - Current LNS is Blind
 24200 Result: One word of Policy information (set by Policy CREAT)
 24250
 †L

00050 .SEC |PAGING|

00100

00150 .SUBSEC |INTRODUCTION|

00200

00250 The single largest impact of the PDP-11 on the design of the paging
00300 system is that the PDP-11 processor is only able to generate a 16-bit
00350 address. Thus user programs, at any instant, may address at most 64K
00400 bytes, or 32K words. The second largest impact arises from the fact
00450 that the relocation hardware divides the user's address space into
00500 eight 8K-byte units called "Page frames". Since this is a rather
00550 small address space, much of the design of the paging system is
00600 oriented toward making these restrictions somewhat easier to live
00650 with.

00700

00750 In the following material we shall use the term "Page" to refer to
00800 an Object, in the HYDRA-technical sense of that word, of type PAGE.
00850 In many contexts the term "Page" may also be read to mean the
00900 information contained in the PAGE Object. The term "Page frame", or
00950 simply "frame", on the other hand, will be used to refer to the area
01000 of physical primary memory (core) in which the information content of
01050 a Page Object resides. The term "frame" is also used to indicate a
01100 portion (1/8th) of the user's address space; context should
01150 disambiguate these uses.

01200

01250 Since Pages are Objects, a user program may, and generally will have
01300 one or more Capabilities which reference specific Pages. These
01350 Capabilities may be in the LNS of an executing LNS or contained in
01400 some Object, e.g., a Directory, which can be named by a Path rooted in
01450 the current LNS. Possession of a Capability for a Page, however, does
01500 not make it addressable. In particular, it is possible that many more
01550 Pages may be named in some particular LNS than can be simultaneously
01600 addressed by the PDP-11 hardware. Thus the paging system defines
01650 means by which the user may specify and alter the set of Page Objects
01700 which are physically present in primary memory and which of these may
01750 be directly accessed at any instant.

01800

01850 Each active LNS has associated with it a CPS (Current Page Set) and
01900 an RPS (Relocation Page Set). The set of pages referenced by the CPS
01950 is guaranteed to be core-resident while the LNS is executing. The set
02000 of pages in the RPS (a subset of those in the CPS) is precisely the
02050 set whose Page frames are named by the relocation hardware of C.mmp
02100 (excluding the stack page which is fixed by the Kernel for the life of
02150 a Process). Thus the Pages in the RPS (plus the stack page) are those
02200 whose information may be accessed directly by instructions executed by
02250 the PDP-11 processor which is executing the user's program. Of
02300 necessity the RPS must refer to seven or fewer pages; no such
02350 restriction exists for the CPS.

02400

02450 Memory allocation (as well as long term scheduling) are controlled
02500 by the particular Policy Subsystem with which the user's Process is
02550 associated. While in principle, the CPS may be of arbitrary size, in
02600 practice it is advantageous for a user to limit the size of her CPS to
02650 make scheduling more likely, though such guarantees depend on the
02700 particular Policy Subsystem.

02750
02800
02850
02900
02950
03000
03050
03100
03150
03200
03250
03300
03350
03400
03450
03500
03550
03600
03650
03700
03750
03800
03850
03900
03950
04000
04050
04100
04150
04200
04250
04300
04350
04400
04450
04500
04550
04600
04650
04700
04750
04800
04850
04900
04950
05000
05050
05100
05150
05200
05250
05300
05350
05400

.SUBSEC |MANIPULATING PAGE SETS|

The Kall CPSLOAD enters pages into the CPS. Loading the current LNS's CPS implies that the designated pages must be brought into core, and the user may assume that they are. In reality however any i/o necessary to make the Pages core- resident is merely requested at this point and a wait-for-i/o-complete, if necessary, is done only when the user requests that a Page be included in his RPS. It should be noted that if a designated CPS slot previously contained a reference to some other Page, that reference is lost and the corresponding Page may become eligible to be swapped out of core, assuming, of course, that the pages are not referenced by the CPS of some other executing LNS.

The Kall RRLOAD provides the user with the ability to move pages from the CPS to the RPS, and hence to be able to reference these Pages directly. As noted above, this operation may imply waiting for the specified Page to become physically resident in primary memory. Once the Page is resident, however, it will remain resident so long as it remains in the CPS and the procedure is active. When the user's Process is stopped, the pages may be swapped out. They are swapped back in when the Process is reSTARTed.

The CPS, RPS, and the functions listed above effectively define a three level memory system - the Pages namable by, or through, the LNS, those named in the CPS, and those named in the RPS. Normally each of these is a subset of the preceding (the exception being that once a Page Capability is loaded into the CPS it may be deleted from the LNS). For the small program, these sets may be identical and the user need not concern herself with the paging system. For larger programs, the user must manage these sets, and the way in which she does so may significantly impact the performance of her program.

.SUBSEC |INITIALIZATION|

An LNS's LCB contains an IPS (Initial Page Set) which specifies how the CPS/RPS is to be initialized when it is Called (by automatically performing CPSLOADs and RRLOADs).

INCPS - Initial size of the CPS

ICPS - 47 words long, the first 'Incps' of which are used to initialize (CPSLOAD) the CPS. Each word contains:

0 - CPS slot will be empty

+m - CPS slot will be CPSLOADed with the Page whose Capability is in the m'th slot of the LNS's C-List

-m - Just like +m, except the Capability is deleted from the LNS's C-List as well (This is useful for pages which the program never manipulates, but must be used carefully, since the Capability may be deleted even if the Call fails)

05450 IRPS - Seven words used to initialize the seven RPS slots. Each word
05500 either contains an index into the CPS (that page will then be
05550 RRLOADED) or 0 (addressing such a page will cause a NXM error - Non
05600 eXistant Memory. The same thing occurs if the CPS slot was empty).

05650
05700 MAXSIZE - Maximum CPS size. Fixed for the life of the LNS.
05750

05800 When an LNS is incarnated from a PROCEDURE, the IPS in the LNS's LCB
05850 is copied from the IPS fields of the PROCEDURE's ICB. Slots in the
05900 ICPS may denote Page Templates in the PROCEDURE's C-List. In the LNS,
05950 these will denote Capabilities for Page Objects passed as arguments in
06000 incarnating the LNS.
06050

06100 When an LNS Calls another LNS, the pages in the Caller's LNS become
06150 eligible to be swapped out. When the Callee returns, the Caller's
06200 pages are automatically first swapped back into core if necessary
06250 before execution proceeds.
06300

06350 An LNS's IPS remains unchanged during the life of the LNS. Hence,
06400 if an LNS KRETURNS and is subsequently LNSCALLED again (or made the
06450 initial LNS of a Process), its CPS and RPS will be re-initialized
06500 using the same IPS, even though the C-List of the LNS may have changed
06550 as a result of previous execution, and even though execution will
06600 continue at the PC following the KRETURN.
06650

06700 Multiple usage of an LNS may of course be prevented by use of the
06750 REUSE Flag in the LCB's PS word (See Subsection on the LCB & ICB)
06800
06850

06900 .SUBSEC |CPS SIZE & THE WORKING SET| 06950

07000 There are 2 limits placed on the size of a CPS. First, the Kernel
07050 has a fixed limit on the total number of CPS slots allocated to active
07100 LNS's (those Called which have not yet Returned) in a Process.
07150 Secondly, a Process's PCB contains a field (CPSMAX) which limits the
07200 maximum CPS size for any LNS executing under the Process. A Call may
07250 fail if the Called LNS's MAXSIZE exceeds the first limit, or if the
07300 LNS's INCPS field exceeds the Process's CPSMAX.
07350

07400 The Kall WORKSET provides a way for (all but Blind) LNS's to
07450 dynamically change the size of the CPS (the LNS's Working Set). It is
07500 always possible (and usually advantageous) to lower the CPS size. It
07550 may not be raised at all above the LNS's MAXSIZE, but it may be raised
07600 over the Process's CPSMAX. If it is, the Process is stopped, and much
07650 like the Kall RUNTIME, the Policy Subsystem is given a chance to raise
07700 the Process's CPSMAX so that the WORKSET Kall will succeed when the
07750 Process is restarted.
07800

07850 A Call or Return always causes a WORKSET to be implicitly executed
07900 since the CPS size may differ in the Caller and Callee. If, on a
07950 Return, CPSMAX is lower than the Caller's CPS size, not only will the
08000 Process be stopped, but it will not be successfully restarted until
08050 CPSMAX is adequately raised (it will just be stopped again). One
08100 small additional point; a Blind LNS may not Call an LNS whose initial

08150 CPS size is greater than the current CPS size.
08200
08250
08300 .SUBSEC |AUXILIARY RIGHTS FOR PAGES|
08350
08400 Two pre-defined auxiliary rights for pages have a somewhat special
08450 property. They are used by the C.mmp hardware when loaded into the
08500 RPS to determine how the page may be addressed by PDP-11 instructions.
08550
08600 A Page loaded from a Capability lacking PGWRTS (or the Kernel right
08650 MDFYRTS) may not be written into.
08700
08750 A Page loaded from a Capability with CACHRTS (and the Kernel right
08800 FRZRTS) is cacheable. The right will be used in conjunction with
08850 the PDP-11 code cache when it is implemented.
08900
08950 In addition, the auxiliary right CPSRTS allows the Page to be
09000 CPSLOADed. If a Page Capability lacks CPSRTS but does contain
09050 COPYRTS, it is called an Initialization Page. The Page may be COPYed,
09100 and the Capability for the COPYed Page will have CPSRTS (as well as
09150 PGWRTS and CACHRTS).
09200
09250
09300 .SUBSEC |COPYING PAGES|
09350
09400 When a PAGE is COPYed, a CPS slot must additionally be specified
09450 indicating where the page may be CPSLOADed. So the COPY Kall for
09500 Pages is specified as follows:
09550
09600 COPY (Nnew, Npage, Ncps) - Copying of Page Object
09650 Parameters:
09700 Nnew - Simple index, Empty
09750 Npage - Simple index, PAGE Object Reference, COPYRTS
09800 Ncps - Positive integer, no greater than the current LNS's
09850 CPS size
09900 Effect: Creates a new Page Object and places a Capability for it
09950 in Nnew. In addition, the contents of the page referenced by
10000 Npage is copied into the new page. The new page is then
10050 CPSLOADed in the Ncps'th CPS slot.
10100 The Kernel rights of the new Capability in Nnew will be the
10150 same as those in Npage plus DLRTS, however, all Auxiliary rights
10200 will be set in Nnew.
10250 Signals:
10300 SCPSBND - Ncps is out of bounds
10350 Result: 0
10400
10450
10500 .SUBSEC |SPECIFICATIONS FOR PAGING KALLS|
10550
10600
10650 PAGE (Path)
10700 Parameters:
10750 Path - Path index; Steps: UCNFRTS,LOADRTS;
10800 Pretarget: STORTS,MDFYRTS; Target: Empty

10850 Effect: Creates a Page Object and places a Capability for it with all
10900 relevant rights but ALLYRTS & FRZRTS in Path's Target.
10950 Result: 0
11000
11050
11100 CPSLOAD (Nlns, <cps-page-pairs>)
11150 Parameters:
11200 Nlns - Simple index, LNS Object Reference, MDFYRTS,SETCBRTS
11250 - or 0
11300 <cps-page-pairs> - One or more pairs of < Ncps, Path >, where:
11350 Ncps - Positive integer, no greater than the LNS's
11400 current CPS size
11450 Path - Path index; Pretarget: LOADRTS;
11500 Target: PAGE Object Reference, CPSRTS
11550 - or 0
11600 Effect: For each pair, loads the Page targeted by Path into
11650 the Ncps'th CPS slot of the LNS denoted by Nlns (the current
11700 LNS if Nlns is 0). If Path is zero, the CPS slot is just
11750 emptied.
11800 See RRLOAD for additional effects.
11850 Signals:
11900 SCPSBND - Some Ncps is out of bounds (above the CPS size or
11950 below 1). SIGDATA contains the index of the bad pair
12000 - The usual signals can occur because of a bad Path
12050 specification. In addition, SPAGE will be or'ed in and
12100 and SIGDATA will contain the index of the bad pair.
12150 Result: 0
12200
12250
12300 RRLOAD (Nlns, Nrps, Ncps)
12350 Parameters:
12400 Nlns - Simple index, LNS Object Reference, MDFYRTS,SETCBRTS
12450 - or 0
12500 Nrps - 1 through 7
12550 Ncps - Positive integer, no greater than LNS's CPS size or 0
12600 Effect: Loads a page into the Nrps'th RPS slot of the LNS denoted
12650 by Nlns (the current LNS if Nlns is 0) from the Ncps'th CPS
12700 slot. If Ncps is zero, the RPS slot will be set to NXM.
12750 If the CPS slot was CPSLOADED from a Capability
12800 with both CACHRTS & FRZRTS, the page may be cached. If the
12850 CPS slot was CPSLOADED from a Capability with
12900 both PGWRTS & MDFYRTS, the page may be written into.
12950 Signals:
13000 SCPSBND - Ncps is out of bounds.
13050 SRPSBND - NRPS is not 1 through 7
13100 Result: CPS slot index of the page previously loaded in the
13150 Nrps'th RPS slot (0 if RPS slot was NXM).
13200
13250
13300 WORKSET (Nlns, Size)
13350 Parameters:
13400 Nlns - Simple index, LNS Objects Reference, MDFYRTS,SETCBRTS
13450 - or 0, in which case, the current LNS must not be Blind
13500 Size - Positive integer, no greater than the LNS's CPS MAXSIZE

13550 Effect: Changes the CPS size of the LNS denoted by Nlns (the current
 13600 LNS if Nlns is 0).
 13650 If Nlns is zero and Size is greater than the current Process's
 13700 CPSMAX, then the Process is stopped. If CPSMAX has not been
 13750 raised to cover Size when the Process is restarted, the Kall
 13800 fails.
 13850 Signals:
 13900 SIPSMAX - Size greater than MAXSIZE.
 13950 SCPSMAX - CPSMAX has not been raised to cover Size. SIGDATA
 14000 contains CPSMAX.
 14050 SBLND - Current LNS is Blind
 14100 Result: 0
 14150
 14200
 14250 Paging Signals for LNSCALL & Process CREAT:
 14300 - The usual signals occur if an ICPS entry denotes something
 14350 other than a Page Object Reference with CPSRTS, however,
 14400 SPAGE will be or'ed with the Signal. SIGDATA will
 14450 contain the bad ICPS index in its low order byte and
 14500 the bad LNS slot is denotes in its upper byte.
 14550 SCPSBND - An IRPS slot contains a bad index into the CPS.
 14600 The low order 3 bits of the signal indicate the
 14650 bad RPS slot (1 - 7).
 14700 SIPSMAX - INCPS is greater than MAXSIZE
 14750 SCPSMAX - One of three things may be wrong:
 14800 1) MAXSIZE > available remaining Process CPS allocation
 14850 2) Current LNS is Blind and INCPS > current CPS size
 14900 3) Current LNS is not Blind and INCPS > CPSMAX even after
 14950 the Process has been stopped and restarted.
 15000 If the current LNS is not Blind, SIGDATA contains CPSMAX
 15050 in its low order byte and the available remaining Process
 15100 CPS allocation in its high order byte.
 15150
 ↑L

00050 .SEC |THE PASSIVE GST|

00100

00150 .SUBSEC |INTRODUCTION|

00200

00250 The collection of Objects is called the GST (Global Symbol Table).
00300 The entire GST is too large to completely reside in main memory. So,
00350 only actively referenced Objects (the Active GST) are kept in core.
00400 The remainder of the GST (the Passive GST) is kept in secondary
00450 memory.
00500

00550

00600 If an Object is in the Passive GST, it will be brought into the
00650 Active GST when it is referenced. Normally, it will migrate back to
00700 the Passive GST when no Capabilities for the Object are in Active
00750 Objects. Though not currently implemented, there will be a limit to
00800 the amount of Active GST space that a Process may use (similar to the
00850 CPS limit, CPSMAX, in the PCB). Thus, it is necessary to allow a
00900 user to PASSIVATE an Object. The Active GST space occupied by the
00950 Object will then no longer be charged against the Process until an LNS
01000 executing under the Process subsequently references the Object. The
01050 Kall PASSIVATE will not actually cause the Object to migrate back to
01100 the Passive GST unless no other processes are actively referencing it.

01150

01200 The Kernel takes great care to insure the reliability of the GST.
01250 For example, if an error occurs in an Active Object due to faulty
01300 memory, the Kernel will attempt to fix it by using available redundant
01350 information in the Object structure as well as the most recent copy of
01400 the Object in the Passive GST. Thus, it is useful to provide a Kall,
01450 UPDATE, that for reliability reasons, updates the most recent copy of
01500 the Object in the Passive GST, regardless of whether or not other
01550 Processes are actively referencing it.

01550

01600

01650 .SUBSEC |SPECIFICATIONS FOR PASSIVE GST KALLS|

01700

01750

01800 PASSIVATE (Path)

01850 Parameters:

01900 Path - Path index; Pretarget: LOADRTS; Target: Defined.

01950 Effect: If Path's Target is last Active reference for
02000 the Object it references, the Object will migrate back to
02050 the Passive GST and each Capability in the Object's C-List
02100 will also be PASSIVATED.

02150 Result: 0

02200

02250

02300 UPDATE (Path)

02350 Parameters:

02400 Path - Path index; Pretarget: LOADRTS; Target: Defined.

02450 Effect: Has the same effect as PASSIVATE, except the Object will
02500 be updated in the Passive GST in any case. In addition,
02550 each Capability in the C-List of the Object referenced is
02600 UPDATED.

02650 Result: 0

02700

↑L

00050 .SEC |PORTS|

00100

00150 The documentation of the port system is being revised. Beware!

00200

00350

00400

00450

00500

00550

00600

00650

00700

00750

The Hydra Message System is the primary means of communication, synchronization and input/output for user PROCESSES. It consists of a set of primitive Kernel Calls which allow PROCESSES to exchange "messages" with each other and with the input/output system via software switching and queueing centers called PORTS. Message transfers are fully synchronized so that other forms of synchronization, i.e., semaphores, mailboxes, etc. will often be unnecessary.

00800

00850

00900

00950

01000

01050

01100

01150

01200

.SUBSEC |WHAT IS A MESSAGE|

01250

01300

01350

A message is basically a string of bytes attached to some routing and queueing information.

01400

01450

01500

More concretely (but not right down at the nitty-gritty) a message has four parts:

01550

01600

01650

01700

01750

01800

01850

01900

01950

02000

02050

- 1) A message "type",
- 2) A "reply stack" (possibly null) of places the message has been sent from and to which it might return as a reply, and
- 3) A text buffer of length ≥ 0 which may be partially or completely filled with information.
- 4) An owner - i.e. the PORT in which the message was originally created and to which the (storage) resources used by the message are charged until the message is destroyed.

02100

02150

02200

02250

02300

02350

02400

The message type is an integer in the range 0-15 (decimal). It is not a static attribute fixed at the time of creation of the message. Instead it is set every time the message is sent (via SEND, RSVP, or REPLY) which

02450

02500

02550

02600

02650

may in general be many times before its destruction. When waiting for a message a PROCESS might choose to accept only those of a given type or a given set of types. Thus the programmer may encode some meaning or classification scheme into his use of the message type field as a convenience in structuring the communication among several PROCESSES.

02700 He might, for example, use the type to distinguish "normal" messages
 02750 from "exceptional" and "catastrophic", or to distinguish replies from
 02800 non-replies.
 02850

02900 Type 0 messages have a special meaning under certain circumstances
 02950 which are discussed later under the description of REPLY. If the
 03000 programmer is not interested in those circumstances he may use type 0
 03050 just as he would any other.
 03100

03150 The "reply stack" of a message is employed when the programmer uses
 03200 the RSVP or
 03250 the REPLY command. It is a stack of places (i.e., PORT, input channel
 03300 pairs) which are eligible to receive replies to this message.
 03350 Basically, the RSVP operation causes a frame of data about
 03400 the sender and the reply he wants to be pushed onto the message's stack w
 **hile

03450 the REPLY operation pops one (or more) frames from the
 03500 stack and uses the information to return the reply. The use
 03550 of this stack is described in greater detail under the descriptions of
 03600 RSVP and REPLY. Here it is impORTant only to note that the maximum
 03650 stack depth (possibly zero) is set at the time of creation of the
 03700 message and is static.
 03750

03800 The text-buffer PORTion of the message is where the data (or text)
 03850 is stored. It has a maximum length decided by the user at MCREATE-
 03900 time and cannot be changed.
 03950 The text buffer may be partially or completely filled using the MWRITE
 04000 command so that the "length" of the message is always less than or
 04050 equal to the length of the buffer. The contents of the text buffer
 04100 of a message are, of course, completely uninterpreted by the Kernel.
 04150 The "meaning" of the message is decided by the communicating PROCESSES.
 04200

04250 It is perfectly legitimate to have a text buffer of length zero
 04300 (no text buffer). If the programmer can communicate all he needs to
 04350 in the type field then there is no need for text at all. The current
 04400 maximum length of a text buffer is 1024 words (decimal).
 04450

04500 The owner of the message is the PORT in which it was originally crea
 **ted.
 04550 At the time a PORT is created it is given an allotment of storage to be u
 **sed
 04600 for the creation of messages. When a message is created the amount of
 04650 storage it uses is deducted from the resource account of the PORT. If t
 **he PORT has
 04700 insufficient resources, the message cannot be created. The resources ar
 **e
 04750 returned to the creating PORT whenever the message is destroyed. The pu
 **rpose
 04800 of this feature is to limit the total number of messages outstanding in t
 **he
 04850 system, thus preventing the disaster that might otherwise be caused if
 **a
 04900 PROCESS tried to create an unbounded number of messages.
 04950

05000 For efficiency reasons messages are not implemented as true
05050 Hydra objects with unique names and capability lists. Consequently
05100 there are no capabilities for them; they cannot be passed as parameters
05150 to PROCEDURES; and they cannot appear in DIRECTORIES. However they are
05200 similar to objects in that they can only be manipulated indirectly
05250 through Kernel Calls and they reside in storage belonging to the Kernel.

05300
05350
05400
05450
05500

05550 .SUBSEC |WHAT IS A PORT|

05600
05650
05700
05750
05800
05850

A PORT is a software post-office where messages are queued,
received, stored and dispatched. Messages may be routed from one PORT
to another (or to the same PORT) or from a PORT to an I/O Device object,
provided that a "connection" has been established first.

05900
05950
06000
06050
06100

Unlike messages, PORTs really are full-fledged Hydra objects in
the technical sense. Furthermore, they are predefined and understood
directly by the Kernel in a way similar to objects of type PAGE, LNS,
PROCEDURE, etc.

06150
06200
06250
06300

A PORT should be thought of as having five main parts:

06350
06400
06450
06500
06550
06600
06650
06700
06750
06800
06850
06900
06950

- 1) A Resource Account - the total amount of storage (in words) allo
**wed
for outstanding messages created in this PORT.
- 2) Input Channel Section: 0-16 (decimal) "input channels" for
queueing incoming messages.
- 3) Output Channel Section: A fixed number of "output channels" each
of which may contain the name of (at most) one PORT or
I/O Device object to which messages can be sent.
- 4) Local Name Section: A fixed number "local names". A local
name is a slot for holding a message which has come to the
attention of some PROCESS (i.e., a newly created or received
message). A message can only be referred to by its local name.

07000
07050
07100
07150

- 5) Waiting PROCESS Section: a queue of
suspended PROCESSEs waiting for messages to arrive.

07200
07250

The actual capacity figures for a PORT are established when it is created
and are fixed for its entire lifetime.

07300
07350
07400
07450

07500 .SUBSUBSEC |OUTPUT CHANNELS, INPUT CHANNELS AND CONNECTIONS|

07550
07600

07650

07700

07750

07800

07850

An output channel, when connected, holds a reference to an input channel of some PORT (possibly the same one the output channel is part of) or a reference to an i/o object. Whenever a message is sent it is sent via some output channel to the place that channel references, and
** thus

07900

07950

08000

08050

at least one output channel is necessary if any messages are to leave the PORT (other than as a reply). Here is no simple upper limit to the number of output channels a PORT may have.

08100

08150

08200

08250

08300

08350

08400

08450

An input channel is simply a message queue. Since all incoming messages are received through an input channel, any PORT which is to receive messages must have at least one. A single PORT may have up to 16 input channels. Multiple input channels can be useful because the RECEIVE routine allows a PROCESS to wait for messages arriving on any subset of input channels. He can thus assign meanings to certain input channels as a convenience in his communication structure.

08500

08550

08600

08650

08700

08750

08800

08850

The CONNECT operation is used to "connect" an output channel to an input channel (or to an I/O Device object). Once a connection is made between two PORTs, messages can be sent between them in the direction of the connection. A connection may be broken using the DISCONNECT operation, and in general connections may be established, broken and then reestablished to somewhere else many times during the lifetime of a PORT (although this is not expected to be frequent).

08900

08950

09000

09050

09100

09150

09200

09250

09300

09350

09400

09450

An output channel can be connected to at most one input channel at a time. However, many output channels may be connected to the same input channel. Thus, when a message is sent via an output channel it is always clear where it is going. But when a message is received from an input channel it is not in general clear which of several places it may have come from unless the programmer restricts himself to a one-to-one connection pattern or labels each connection with a "connection ID". (See CONNID parameter in the CONNECT operation.)
It is not possible to tell how many, if any, output channels are connected to a particular input channel.

09500

09550

09600

09650

09700

09750

09800

09850

09900

09950

10000

A brief bit of Hydra philosophy might be injected here. Notice first that messages are sent from PORT to PORT, not PROCESS to PROCESS. Therefore, one PROCESS need not know the name of (i.e., have a capability for) another PROCESS to get a message to it. This is especially important in a system of several equivalent server PROCESSES which are sharing a message PROCESSING load. Merely sending a message to the PORT that they presumably share is sufficient to assure that one of them (and only one) will receive it. The number of server PROCESSES may change dynamically with time with no effect upon the action of the requesting PROCESSES.

10050

10100

10150

10200

10250

Another consequence of this Message System design is that the programmer of a system using PORTs has strong control over the communication structure and can use the capability mechanisms of Hydra enforce that control. Messages cannot be sent arbitrarily between any two PORTs - only between PORTs that are connected. By appropriately

10300 controlling the flow of capabilities for PORTs, particularly those
10350 with right PCONNRTS of connection and disconnection, he
10400 can assure the integrity of the connection graph. He can further
10450 restricts his communication
10500 by limiting the distribution of the other auxilliary rights for the
10550 message-handling primitives, thereby achieving further protection.
10600 (See the list of auxilliary rights supported by the
10650 Message System.)

10700
10750
10800
10850
10900

10950 .SUBSUBSEC |LOCAL NAMES|

11000
11050

11100 Every PORT contains a set of message-holding pigeon-holes
11150 called "local names" which are numbered from 0.
11200 There is no simple upper limit to the number of local names
11250 a PORT may have.
11300 Each such local name can hold only one message at a time. In order
11350 for a PROCESS to perform any of the primitive operations upon a message,
11400 that message must be sitting in a local name of some PORT.

11450
11500

11550 When referring to a message in order to perform an operation on
11600 it the user cannot simply give its address because he has no way of
11650 getting it (or accessing it even if he had it). Instead he refers to
the message by specifying the pair (P,L) where P is the LNS index of a ca
**pability for

11700 a PORT and L is the index of a local name within that PORT. (We
11750 will abbreviate from now on and say that L is a local name in some
11800 PORT, as opposed to the index of a local name.) Each of the
11850 primitive operations MREAD, MWRITE, SEND, RSVP and REPLY have just such a
11900 pair as their first two arguments.

11950

12000 A local name is in one of two states, "full" or "free", according
12050 to whether it holds a message at the moment or not. When a message
12100 is created via MCREATE the system searches for a "free" local name and
12150 allocates it to the new message, changing the state of the local name
12200 to "full". The user can then operate the message using MWRITE, SEND, RSV
**p

12250 or REPLY. Once SEND, RSVP or REPLY is done, the local name becomes
12300 "free" again. Similarly, when a message is received via RECEIVE, the
12350 system has to search for a free local name to put it in before
12400 returning to the user, whereupon he may perform MREAD, etc., on the
12450 message.

12500

12550 If the Message System is unable to find a "free" local name an
12600 error condition is signaled. (NOTICE: The PROCESS is NOT suspended.
12650 This is to avoid deadlock in the case that only one PORT is using
12700 the PORT.) Thus, the local names of a PORT should
12750 be considered a valuable and scarce resource. If a PROCESS or group of
12800 PROCESSES uses the local names of a PORT unwisely it will require very
12850 complex algorithms to properly handle the error signals and get out of

12900 the jam without deadlock or other disaster. It may be advisable for
12950 PROCESSES sharing a PORT to control their use of local names via some
13000 kind of limit semaphore. However, any such arrangement is outside
13050 the Message System.

13100
13150

13200 A single PORT may have up to 64 local names. the exact number is
13250 decided at the time the PORT is created and is static for the life of
13300 the PORT. Since, in order to do any message operations a local name
13350 is required, every PORT must have at least one. For simple message
13400 PROCESSING, where each PROCESS disposes of one message before beginning
13450 to PROCESS another, no more than one local name per PROCESS using the
13500 PORT is necessary.

13550

13600 "Local names" are so called because they are "local" to a single
13650 PORT. However, if several PROCESSES are using the same PORT it is
13700 possible for one PROCESS to interfere with another by operating on
13750 messages in local names that were never assigned to that PROCESS by
13800 MCREATE or RECEIVE. In that sense local names are really
13850 "common" or "global" to all PROCESSES using the same PORT. It is
13900 therefore very important that PROCESSES using the same PORT
13950 cooperate with one another in this respect.

14000

14050

14100

14150

14200

14250 .SUBSUBSEC |WAITING PROCESSES|

14300

14350

14400

14450

14500

14550

14600

14650

14700

14750

14800

14850

14900

14950

15000

15050

15100

15150

15200

15250

15300

15350

15400

15450

15500

15550

.SUBSUBSEC |RELATION OF PORTS TO I/O OBJECTS|

As previously described, an output channel of a PORT may optionally
be connected to an input/output device object instead of to an input
channel of a genuine PORT. The device object, though technically not
part of the Message System, acts abstractly as though it were

15600 really a PORT with one input channel and no output channels. An i/o
15650 request for the physical device associated with the device object is
15700 then implemented as a message sent to the device object. The result
15750 of the i/o operation is implemented as a reply to the request message.
15800 Exceptional and normal replies will generally have different types and
15850 thus return to different places according to the reply stack of the
15900 request message. (Historically, the requirement for exception handling
15950 in i/o was the primary model for the RSVP/REPLY mechanism of the
16000 Hydra Message System.)
16050
16100

The fact that a device object viewed as a PORT has no output channel
**s

16150 means that there can never be a CONNECT operation between two device
16200 objects. It also means that the i/o system never creates or sends
16250 a message. It can only reply to messages that have been sent to it.
16300

16350 There is one departure from the abstraction that a device object
16400 acts like a PORT: only one output channel at a time can be connected
16450 to any particular device object. This corresponds to the notion that
16500 - at least at the lowest level - a hardware device belongs to only one
16550 PROCESS at a time.
16600
16650
16700
16750

CONNECT (Port1, Outchan, Port2, Inchan, Connid)

Parameters:

16800
16850 PORT1 - Simple Index of PORT object reference;CNFRTS;PCONNRTS
16900 Outchan - Integer, either -1 or between 0 and N-1 inclusive,
16950 where N is the number of output channels in the first
17000 PORT.
17050 Port2 - Simple Index of PORT object or I/O Device object;
17100 PCONNRTS
17150 Inchan - Integer between 0 and N-1 inclusive, where N is the
17200 number of input channels in the second PORT. This
17250 parameter is ignored if Port2 refers to an I/O Device
17300 object.
17350 Connid - Any 16 bit pattern.
17400
17450

17500 Effect: The output channel designated by Outchan in the first PORT
17550 is "connected" to the input channel designated by Inchan in the
17600 second PORT, thereby forming a path for messages to travel. The
17650 output channel is marked "connected" so that further CONNECT
17700 operations on the same output channel will fail until
17750 and unless it is DISCONNECTed first.

17800 If Outchan is -1 the kernel selects a free output channel
17850 and makes the connection, signalling if there are no free
17900 output channels.

17950 Connid is used as a symbol to identify the connection, and
18000 is part of the information stamped on every message that travels
18050 along the path made by the connection. It may be used for any
18100 purpose since it is completely uninterpreted by
18150 the kernel. (See RECEIVE for another reference to this feature.
**)

18200 For purposes of the CONNECT operation an I/O Device object
18250 is identical to a PORT which is limited to only one input
18300 channel. However, there can be no more than one connection
18350 to an I/O Device object. A signal will be generated if
18400 an attempt is made to connect to an I/O Device object which
18450 is already connected. (There is no such restriction on
18500 connections to the input channels of a PORT.)
18550

18600 Signals: All signals from CONNECT will have SGPCONNECT in bits
18650 6-10 and one of the following values in bits 0-5:

18700
18750 SGPOCHANRANGE - Outchan is less than -1 or greater than
18800 highest output channel index of PORT1.
18850 SGPNOFREECHAN - Outchan is -1, but there are no free
18900 output channels available.
18950 SGPALREADYCONNECTED - Outchan specifies an output channel
19000 which is already connected.
19050 SGPICHANRANGE - Inchan is negative, or greater than the
19100 highest input channel index of in Port2.
19150 SGPIOERR - Attempted connection to an I/O Device
19200 object which is already connected.
19250

19300 Result: CONNECT normally returns the index of the output channel
19350 which was connected. This is either Outchan or, in the
19400 case Outchan is -1, the selected output channel.
19450

19500
19550
19600

DISCONNECT (Port, Outchan)

19650 Parameters:

19700 PORT - Simple Index of a PORT object; CNFRTS;PCONNRTS
19750 Outchan - Integer index of the output channel to be disconnected.
19800

19850

19900 Effect: The output channel Outchan of the specified PORT is
19950 logically "disconnected" from wherever it was "connected".
20000 The output channel may now be re-connected to somewhere else.
20050 There is no distinction between disconnecting from a PORT and
20100 disconnecting from an I/O Device object.
20150

20200 Signals: Signals from DISCONNECT have SGPDISCONNECT in bits 6-10
20250 and one of the following values in bits 0-5:

20300
20350 SGPOCHANRANGE - Outchan is negative or larger than the
20400 largest output channel index in the PORT.
20450
20500 SGPUNCONNECTED - The output channel is not connected and
20550 thus cannot be disconnected.
20600

20650 Result: 0

20700
20750
20800
20850

20900 MCREATE (Port, Bufflength, Stackdepth)

20950 Parameters:

21000

21050

PORT - Simple Index of a PORT object; CNFRTS; MCREATERTS

21100

Bufflength - Integer between 0 and #4000 (octal); specifies the length of the message buffer in bytes, i.e. the maximum length of the text of the message.

21150

21200

21250

Stackdepth - Integer between 0 and 10 (decimal) inclusive; specifies the maximum depth of the message's reply stack.

21300

21350

21400

21450

Effect: A new message is created according to the specification of of the Bufflength and Stackdepth parameters. A free local

21500

21550

name is found in the PORT and the new message is assigned to

21600

that local name. The resources (storage) consumed by the message are deducted from the resource account associated with the PORT.

21650

21700

21750

21800

Signals: All signals from MCREATE have SGMCREATE in bits 6-10 and one of the following in bits 0-5:

21850

21900

21950

SGMBUFFLENGTH - Bufflength is negative or greater than the implementation defined maximum of #4000 bytes.

22000

22050

SGMSTACKDEPTH - Stackdepth is negative or greater than the implementation defined maximum of 10.

22100

22150

SGMRESOURCES - There are insufficient resources left in the resource account associated with the PORT to allow creation of this message.

22200

22250

22300

SGMNOFREELNAME - There are no unassigned local names to give to the message.

22350

22400

22450

Result: MCREATE normally returns the local name assigned to the new message.

22500

22550

22600

22650

22700

22750

MREAD (Port, Lname, Pos, Len, Textadr)

22800

Parameters:

22850

PORT - Simple Index of a PORT object; CNFRTS; MREADRTS

22900

Lname - Integer local name in the PORT

22950

Pos - Byte index (origin 0) of the section of the message buffer to be read.

23000

23050

Len - Length in bytes of the section of the message buffer to be transferred.

23100

23150

Textadr - Legitimate Stack Memory Address of an area at least Len bytes long.

23200

23250

23300

Effect: The section of the message buffer designated by Pos and Len is copied into the stack area pointed to by Textadr.

23350

23400

23450

Signals: All signals from MREAD have SGMREAD in bits 6-10 and one of the following in bits 0-5:

23500

23550

23600 SGMLNAMERANGE - Lname is negative or out of range of the local
23650 names of the PORT.
23700 SGMLNAMEFREE - Local name Lname is free, i.e. has no message
23750 assigned to it.
23800 SGMBUFFBOUNDS - Pos and/or Len do not specify a segment wholly
23850 contained within the text of the message.
23900 SGMTEXTADR - Textadr does not specify a Legitimate Stack
23950 Memory Address of an area at least Len bytes
24000 long (or the area is not wholly contained in the
24050 legitimate area of the stack).
24100

24150 Result: 0
24200
24250
24300
24350

24400 MWRITE (Port, Lname, Pos, Len, Textadr)

24450 Parameters:

24500 PORT - Simple Index of a PORT object; CNFRTS;MWRITERTS
24550 Lname - Integer local name in the PORT
24600 Pos - Byte index (origin 0) of the section of the message
24650 buffer to be written.
24700 Len - Length in bytes of the section of the message
24750 buffer to be written.
24800 Textadr - Legitimate Stack Memory Address of an area at least
24850 Len bytes long containing the data to be written
24900 into the message.
24950

25000 Effect: The data in the area pointed to by Textadr is copied into
25050 the section of the message buffer specified by Pos and Len.
25100

25150 Signals: All signals from MWRITE have SGMWRITE in bits 6-10 and one
25200 of the following in bits 0-5:

25250
25300 SGMLNAMERANGE - Lname is negative or out of range of the local
25350 names of the PORT.
25400 SGMLNAMEFREE - Local name Lname is free, i.e. has no message
25450 assigned to it.
25500 SGMBUFFBOUNDS - Pos and/or Len do not specify a segment wholly
25550 contained within the message buffer.
25600 SGMTEXTADR - Textadr does not specify a Legitimate Stack
25650 Memory Address of an area at least Len bytes
25700 long (or the area is not wholly contained in the
25750 legitimate area of the stack).
25800

25850 Result: 0
25900
25950
26000
26050

26100 SEND (Port, Lname, Type, Outchan)

26150 Parameters:

26200 PORT - Simple Index of a PORT object; CNFRTS; SENDRTS
26250 Lname - Integer local name of the message to be sent.

26300 Type - Integer in the range 0-15 to become the new type of
26350 the message.
26400 Outchan - Output channel index specifying the destination of the
26450 message.
26500

26550 Effect: The type indicator of the message with local name Lname is
26600 set to Type and the message is sent to the PORT or I/O Device
26650 to which output channel Outchan is connected. Local name
26700 Lname becomes free. There is no effect upon the other
26750 attributes of the message, i.e. its owning PORT, its message
26800 buffer, or its reply stack.
26850 When the message arrives at the destination PORT and input
26900 channel it may satisfy the requirements of one or more PROCESSES
26950 that were blocked in a RECEIVE operation. If so, exactly one
27000 of the eligible blocked PROCESSES is awakened to receive the
27050 message; the other PROCESSES remain blocked. The longest
27100 blocked eligible PROCESS is always selected in order to
27150 enforce a policy of fairness. (Strictly speaking, the PROCESS
27200 is not awakened; rather the appropriate POLICY object is
27250 notified that it may schedule the selected PROCESS.)
27300 If no PROCESSES are blocked at the destination PORT
27350 or if the incoming message does not satisfy the type or input
27400 channel criteria of any of the blocked PROCESSES, then the
27450 message is enqueued (in FIFO order) in the proper input channel
27500 and type queues. It will be received by the first PROCESS
27550 which does a RECEIVE operation on the same PORT for some class of
27600 messages to which this one belongs. Under no circumstances does
 ** the
27650 sending PROCESS get blocked.
27700 If the destination of the message is an I/O Device (as
27750 opposed to a PORT) the I/O system immediately receives the
27800 message and begins to act on it.
27850

27900 Signals: All signals from SEND have SGSEND in bits 6-10 and one
27950 of the following in bits 0-5:
28000

28050 SGMLNAMERANGE - Lname is negative or out of range for this PORT.
28100 SGMLNAMEFREE - Local name Lname is free, i.e. assigned to no
28150 message.
28200 SGMOCHANRANGE - Outchan is negative or out of range for this
28250 PORT.
28300 SGMUNCONNECTED - Output channel Outchan is not connected.
28350 SGMTYPERANGE - Type is not in the range 0-15 inclusive.
28400

28450 Result: 0
28500
28550
28600
28650

28700 RSVP (Port, Lname, Type, Outchan, Messid, Inchan, Replymask)
28750 Parameters:
28800 PORT - Simple Index of a PORT object; CNFRTS; SENDRTS
28850 Lname - Integer local name of the message to be sent.
28900 Type - Integer in the range 0-15 to become the new type

28950 of the message.

29000 Outchan - Output channel index specifying the destination of
29050 the message.

29100 Messid - 16 bit identifier for the message.

29150 Inchan - Integer index of the input channel through which the
29200 reply (if it returns at all to this PORT) is to return
**n.

29250 Replymask - 16 bit mask specifying (with 1-bits) which types of
29300 reply are to return to this PORT. Replies of other t
**types will
bypass this PORT.

29350

29400

29450 Effect: RSVP does the same thing as SEND, but in addition requires
29500 that a reply be generated. The first four parameters to RSVP ar
**e
interpreted exactly like the four parameters to SEND. It is the
29550 last three parameters which provide the information necessary for
29600 ** the
REPLY mechanism and which distinguish RSVP from SEND.

29650 Just before doing the equivalent of a SEND operation, RSVP
29700 pushes a frame of information onto the message's reply-stack.
29750 This frame controls the action of the subsequent REPLY operation,
29800 and includes as data the last three parameters to RSVP: Messid,
29850 Inchan and Replymask.
29900 Rsvp guarentees that a reply message will be generated by som
29950 **eone
at some later time. But it does not guarentee that the reply
30000 will return to the PORT from which the corresponding RSVP was
30050 done. Whether or not a reply is ever received at the PORT wher
30100 **e
the original RSVP was done depends on two things: 1) the
30150 Replymask parameter to RSVP, and 2) the type assigned to the
30200 message at the time the REPLY operation is done (usually by some
30250 other PROCESS.)
30300 If the bit in Replymask corresponding to the type of the mess
30350 **age
is 1, then the reply will be received at the PORT from which the
30400 **RSVP
was done; if not, the PORT from which the RSVP was done will be
30450 bypassed during the REPLY operation and some other PORT (or none)
30500 will receive the reply. Thus, the only way to guarentee that a
30550 reply will be received at the PORT where the RSVP was done is
30600 to specify a Replymask of #177777 (octal). Then the PORT cannot
30650 be bypassed no matter what type is assigned to the message at the
30700 time the REPLY operation is done. (See REPLY for more details.)
30750 A reply to an RSVP-message may or may not return to the
30800 originating PORT, but if it does, it must arrive through an
30850 input channel. The Inchan parameter allows the sender of an
30900 RSVP to specify which input channel any reply will return to.
30950 By turning on bit number Inchan in the channel-mask of a
31000 subsequent RECEIVE operation, the user can receive the reply.
31050 In some applications it is essential to be able to keep track
31100 ** of
individual messages and associate replies with the original rsvp.
31150

31200 The Messid parameter allows this bookkeeping to be done reliably.
 31250 Whatever argument is passed as Messid is used as a "name"
 31300 which stays with the message until the reply is received. When
 **a
 31350 reply is received the original Messid is returned as part of the
 31400 message description. (See RECEIVE for more information.)
 31450 The Messid parameter is completely uninterpreted by the Kernel, s
 **o the
 31500 user is permitted to devise any bookkeeping system he wishes (or
 **none.)
 31550 There is no way that any subsequent handling of the message can
 31600 disturb this identification.
 31650 For more information related to RSVP, see the descriptions of
 31700 SEND, REPLY and RECEIVE.
 31750
 31800 Signals: All signals from RSVP have SGRSVP in bits 6-10 and one
 31850 of the following in bits 0-5:
 31900
 31950 SGMLNAMERANGE - Lname is negative or out of range for this PORT.
 32000 SGMLNAMEFREE - Local name Lname is free, i.e. assigned to no
 32050 message.
 32100 SGMCHANRANGE - Outchan is negative or out of range for this
 32150 PORT.
 32200 SGMUNCONNECTED - Output channel Outchan is not connected.
 32250 SGMTYPERANGE - Type is not in the range 0-15 inclusive.
 32300 SGMICHANRANGE - Inchan is negative or out of range for this
 32350 PORT.
 32400 SGMSTACKOVFL - Reply stack overflow; no more room in the reply
 32450 stack of this message.
 32500
 32550 Result: 0
 32600
 32650
 32700
 32750
 32800 REPLY (Port, Lname, Type)
 32850 Parameters:
 32900 PORT - Simple Index of a PORT object; CNFRTS; REPRTS
 32950 Lname - Integer index (local name) of the message to be REPLYed.
 33000 Type - Type to be assigned to the message.
 33050
 33100 Effect: The REPLY operation is used to delete a message or to return
 **it
 33150 to some PORT where a previous RSVP operation was done to the mess
 **age.
 33200 A record of those PORTs where an RSVP was done to the message and
 ** the
 33250 criteria for receipt of a reply at those PORTs
 33300 is carried around with the message in its reply-stack. Each tim
 **e an
 33350 RSVP is done to the message one stack frame is pushed onto the
 33400 message's reply-stack, and each time a REPLY operation is done, o
 **ne
 33450 or more frames are popped from the reply-stack. Thus, at any

33500 given instant the reply-stack contains frames corresponding to
33550 exactly those PORTs which are be eligible to receive replies.
33600 The REPLY operation proceeds in detail as follows:
33650
33700 1) The value of the parameter Type is assigned to be the
33750 type of the message with local name Lname.
33800
33850 2) Each reply-stack frame in the message is examined,
33900 starting naturally from the stack-top, to see if the c
**urrent
33950 message is among those that were specified in the Repl
**ymask
34000 parameter to the original RSVP operation. (See RSVP.)
34050
34100 3) If not, the reply-stack frame is popped and the examin
**ation
34150 of frames continues. The PORT associated with the po
**pped
34200 frame is "bypassed" and never receives a reply.
34250
34300 4) If so, however, the examination of frames stops. The
**
34350 message is "sent" to the PORT associated with the repl
**y-
34400 stack frame through the input channel specified in the
34450 Inchan parameter to the original RSVP operation. (Se
**e RSVP.)
34500 There the message will either be enqueued or
34550 will be immediately received by a blocked PROCESS, jus
**t as
34600 if the message had been sent using SEND. (The last
34650 reply-stack frame examined is also popped.)
34700
34750 5) If all frames are popped without finding a PORT eligib
**le to
34800 receive the reply, then the message is destroyed. Th
**is is
34850 the only way a message can be deleted under Hydra; the
**re
34900 is no MDELETE Kall.
34950
35000 Signals: All signals from REPLY have SGREPLY in bits 6-10 and one
35050 of the following in bits 0-5:
35100
35150 SGMLNAMERANGE - Lname parameter is negative or out of range for t
**his
35200 PORT.
35250 SGMLNAMEFREE - Local name Lname is free, i.e. is assigned to no
35300 message.
35350 SGMTYPERANGE - Parameter Type is not in the range 0-15.
35400
35450 Result: 0
35500
35550

```

35600
35650
35700 RECEIVE ( Port, Cond, Waitclass, Mask, Descr )
35750 Parameters:
35800 PORT - Simple Index of PORT object; CNFRTS; MRECRS
35850 Cond - Boolean; true if RECEIVE is conditional, i.e. blockin
          **g
35900         not allowed; false if RECEIVE is unconditional and b1
          **ocking
35950         is permitted.
36000 Waitclass - Boolean; true if specifying messages by input channel
          **s;
36050         false if specifying messages by type.
36100 Mask - 16 bit mask specifying either a set of input channels
36150         or a set of types (depending on the Waitclass paramet
          **er.)
36200         Bits are numbered 0-15 from least to most significant
          **.
36250 Descr - Legitimate Stack Memory Address of an area at least s
          **ix
36300         words; RECEIVE fills this area with a description of
          ** the
36350         received message. (See format below.)
36400
36450 Effect: RECEIVE is the basic message-receive primitive of the PORT sys
          **tem.
36500 The user passes a description of the class of messages he wishes
          **to
36550 receive, and the Kernel either immediately returns access to such
          ** a
36600 message, or it blocks the PROCESS until such a message is availab
          **le.
36650 If a message is received, a more detailed description of it
36700 is placed in the user's stack area at Descr so that
36750 he may know what kind of message he has received.
36800 The events in more detail are as follows:
36850 The two parameters Waitclass and Mask form the description of
          **
36900 the class of messages the user wishes to receive. He may either
36950 receive a message which has one of a set of message types, or he
37000 may elect to receive a message that arrives via any one of a set
37050 of input channels. The choice between type-specification and ch
          **annel-
37100 specification is made through the Boolean parameter Waitclass.
37150 The set of channels or types is specified by the parameter Ma
          **sk.
37200 Bits 0-15 of the mask specify either channels 0-15 or types 0-15
37250 (depending on Waitclass.) Thus, if Waitclass = 1 and Mask = #03
          **0777
37300 then only a message which arrives through one of the channels 0-8
37350 or 12-13 will be received. Any one-bits in Mask which correspo
          **nd
37400 to channel indices greater than those allowed for the PORT in
37450 question are ignored.

```


37500 The Waitclass and Mask parameters form a description of a cla
 **ss
 37550 of messages but do not specify a particular message. Thus, ther
 **e may be many
 37600 messages enqueued which fit the description at the time a
 37650 RECEIVE is done. The user has no control over which of the elig
 **ible
 37700 messages will be received beyond what have already been described
 ** under
 37750 the Waitclass and Mask parameters. In particular, he has no way
 **of
 37800 giving "priority" to certain channels or types. Messages are
 37850 selected by the Kernel for receipt subject to only two
 37900 restrictions:

- 37950
 38000 1) Messages will be received in FIFO order within any giv
 **en
 type or any given input channel.
 38050 2) Type and channel queues will be scanned according to a
 38100 "fair" policy, so that no input channels or types will
 38150 be systematically ignored across many RECEIVE operatio
 38200 **ns.

38250
 38300 The Cond parameter specifies whether or not the RECEIVE opera
 **tion
 38350 is "conditional", i.e. whether or not the PROCESS doing the RECEI
 **VE
 38400 is permitted to block. If Cond is true (odd) then no blocking i
 **s
 38450 permitted. Thus, if a message fitting the Waitclass-Mask
 38500 description is available, it will be received; if not, no message
 ** will
 38550 be received, and a signal will be generated.
 38600 However, if Cond is false (even) then blocking is permitted.
 38650 If no satisfactory message is available the PROCESS will be suspe
 **nded
 38700 until one arrives. (Actually the Kernel
 38750 doesn't "suspend" the PROCESS; it stops the PROCESS and
 38800 notifies the POLICY system not to reschedule it until further
 38850 notice. An erroneous POLICY system may schedule the PROCESS
 38900 anyway, but the Kernel will immediately re-stop it and once again
 38950 notify the POLICY system not to reschedule it.)

39000 When a message is received a detailed description of the
 39050 message is is placed in the six-word area that the user provides
 39100 through the parameter Descr. The format of this
 39150 six word area, and the interpretation of the fields are as follow
 **s:

39200
 39250
 39300 -----
 ! LNAME !

 39350 !R! ! TYPE !INCHAN !

 39400 ! LENGTH !
 39450
 39500

```

39550 -----
39600 !          BUFFLENGTH          !
39650 -----
39700 !          MESSID              !
39750 -----
39800 !          CONNID              !
39850 -----
39900
39950 LNAME      - The local name assigned to the received mess
                **age.
40000 R        - Reply-bit: 1 if the message is a reply to an
40050                earlier RSVP; 0 if it is a normal unsolicited
40100                arriving message.  This field is the only
40150                way to distinguish replies from non-replies.
40200 LENGTH    - The length (in bytes) of the text in the mes
                **sage
40250                buffer.
40300 BUFFLENGTH - The length (in bytes) of the message buffer.
                ** Must
40350                be greater than or equal to LENGTH.
40400 MESSID    - If this message is a reply, MESSID contains
40450                the message-id assigned to this message at t
                **he
40500                time the RSVP was done.  (See RSVP.)
40550 CONNID    - If this message is not a reply, CONNID conta
                **ins
40600                the connection-id of the connection through
                **which
40650                the message arrived.  This gives the receiv
                **er of
40700                a message some idea of where the message cam
                **e
40750                from.  (See CONNECT for a discussion of
40800                the idea of a connection-id. )
40850
40900 Signals:  All signals from RECEIVE have SGMRECEIVE in bits 6-10 and
40950                one of the following vales in bits 0-5:
41000
41050 SGMNOFREELNAME - Lname is negative or out of range for this PORT.
41100 SGMPACKADR    - Packadr is not a Legitimate Stack Memory Address
                ** of
41150                a six word area.
41200 SGMCONDRECFAIL - The Cond parameter indicates a conditional recei
                **ve,
41250                but no satisfactory message is available.
41300
41350 Result: RECEIVE normally returns the local name assigned to the received
41400 message.
↑L

```

00050 .SEC |User I/O Operations|

00100

00150 .SUBSEC |Overview from a Subsystem Builder's Viewpoint|

00200

00250 In order to perform input/output operations, the subsystem must
00300 connect a port to an i/o device. This action is performed by means
00350 of the message system's PCONNECT operation, described in [ref].

00400 After a connection has been established successfully, the i/o device

00450 identified by the specified object is available for exclusive use

00500 through the given port and output channel, and such exclusive access

00550 remains effective until disconnection (see PDISCONNECT). All future

00600 operations specify the i/o device indirectly, by way of the port and

00650 output channel to which it is connected, and the i/o device object

00700 is of no further use.

00750

00800 The i/o device object may also be used to request reconfiguration,

00850 but this is a specialized use which is documented in a separate section

00900 ([ref]).

00950

01000 .SUBSEC |Overview from a User Program's Viewpoint|

01050

01100 A user program performs i/o operations in exactly the same

01150 manner as it sends messages via the message system (see [ref]). In

01200 fact, there is no way to determine whether an output connection

01250 is to an i/o device or to another port. A user program merely sends

01300 messages of a prescribed format (see [ref]) and waits for a reply,

01350 if appropriate. The information in the message specifies the requested

01400 operation, and the reply type indicates the outcome of the request.

01450 All message system primitives for sending messages and obtaining

01500 replies are equally applicable to i/o requests.

01550

01600 .SUBSEC |Conventions|

01650

01700 All i/o messages (henceforth referred to as requests) contain at

01750 least an operation code indicating the specific action to be taken.

01800 Most requests also include a buffer, a byte count, and some device

01850 parameters (e.g. a sector address for a disk transfer). This section

01900 outlines the conventions which govern the format of i/o requests,

01950 leaving details of specific operations for the next section.

02000

02050 The operation code is the first word of every i/o message. It

02100 is subdivided into three fields: optype, opcode, and offormat.

02150 The optype places the request into one of four general categories.

02200 Immediate operations require no action by the device itself. Control

02250 operations affect the device, but no data transfer occurs (e.g. tape

02300 rewind). Input operations transfer one or more bytes of data from

02350 the device to memory; output operations transfer data from memory to

02400 the device.

02450

02500 The opcode field determines the particular operation to be

02550 performed within a given class. For many devices, only one operation

02600 of each class will be defined; however, some devices may have several.

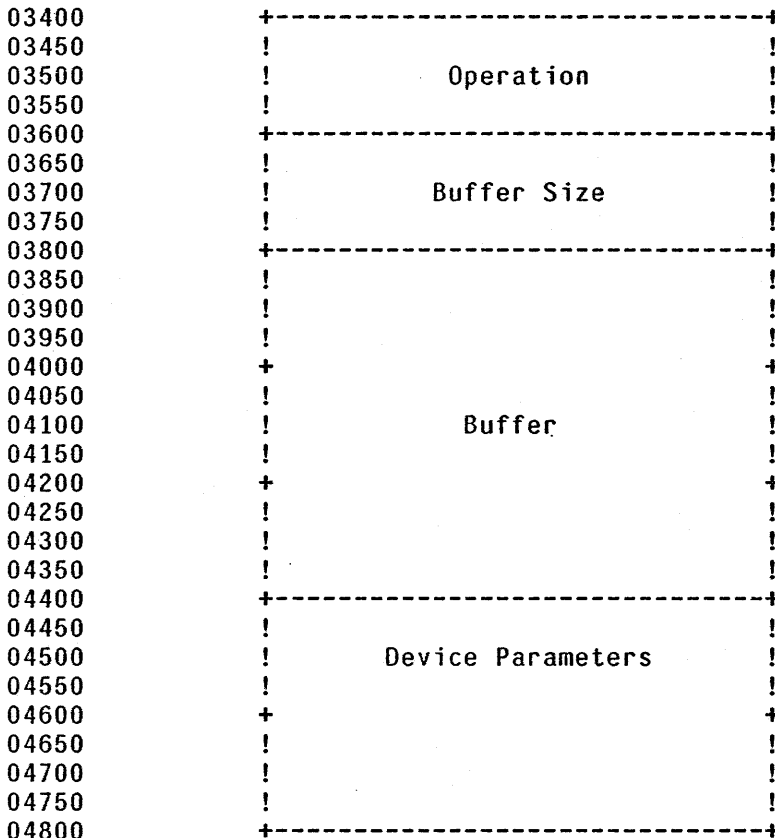
02650 For example, a DECTape has two control operations, rewind and findblock.

02700 The optype and opcode fields together define a unique logical operation,

02750 which may correspond to zero, one, or more physical operations on the
02800 device.
02850

02900 The opformat field provides format information about the i/o request
02950 itself and does not directly influence the operation. It is broken
03000 down into individual bits which specify the existence or nature of
03050 other fields in the request. Not all of these bits may be relevant
03100 to a particular operation, and some operations may outlaw
03150 certain format settings -- consult the descriptions of the specific
03200 actions for details.
03250

03300 The general i/o request assumes the following form:
03350



04850
04900 The operation field has already been discussed. The buffer size field
04950 is normally required only for transfer operations, and holds the
05000 number of bytes of data to be transferred. Some devices (e.g. teletype)
05050 allow the buffer size field to be omitted on some transfer operations;
05100 in such cases the omission is indicated by a bit in the opformat field.
05150 The buffer area is of the size specified by the byte count and is
05200 required for all operations which transfer data. The buffer is normally
05250 contained within the message itself, but may be specified indirectly
05300 as an address within the requesting lns's address space (cps --
05350 see [ref]). In this case, a format bit is set in the opformat field
05400 and the buffer address is a two word quantity whose first word is a

05450 cps index and whose second word is a 13-bit displacement. The device
05500 parameters field is operation dependent and for sequential devices is
05550 usually omitted. It frequently contains positioning information for
05600 read/write heads, but may specify auxiliary information for any of
05650 the four optype classes.
05700
05750 The outcome of i/o requests is reported via the message system
05800 message type [ref], which summarizes the result of the operation. If
05850 additional information is necessary to define the outcome, it will be
05900 appended to the message following the last word supplied by the requestin
**g
05950 process. No information in the message itself (except possibly the
06000 buffer during an input operation) is ever altered during an i/o operation
**.

06050 Thus the contents of a failing request may be examined to determine
06100 the cause of the error. A single type, OPDONETYPE, indicates a
06150 successful completion, while other reply types are used to
06200 denote errors. The specific reply codes are discussed later.
06250
06300 .SUBSEC [Specific Device Operations]
06350
06400 This section describes the operations which are permitted for
06450 each of the several device classes supported. It should be noted that
06500 the values for specific fields are given symbolically rather than as
06550 absolute numeric quantities. The equivalences are established by
06600 use of the BLISS/11 "require" file UIO.REQ[N810HY00], which should always
** be used

06650 by user programs.
06700
06750 1) Operations common to all devices
06800
06850 A limited number of operations are defined to have a
06900 common action for all devices.
06950
07000 a) DIDENTIFY
07050
07100 Class: Immediate
07150 Format restrictions: not applicable
07200 Byte Count: not used
07250 Buffer: not used
07300 Device Parameters: not used
07350 Other Information: returns static information
07400 pertaining to the device in the words immediately
07450 following the operation code as follows:
07500

07550	+	-----	+	-----	+
07600	!		!		!
07650	!	PNUM	!	CTYPE	!
07700	!		!		!
07750	+	-----	+	-----	+
07800	!		!		!
07850	!	Registers Address	!		!
07900	!		!		!
07950	+	-----	+	-----	+

```

08000      !                               !
08050      !   Interrupt Vector Address   !
08100      !                               !
08150      +-----+-----+
08200      !// // // // // // // // // // !
08250      !// // // // // // // // // // !   Unit Number
08300      !// // // // // // // // // // !
08350      +-----+-----+
08400
08450      CTYPE   CONTROLLER TYPE
08500      PNUM    PROCESSOR NUMBER
08550

```

b) DSTATUS

```

08600
08650      Class: Control
08700      Format restrictions: not applicable
08750      Byte Count: not used
08800      Buffer: not used
08850      Device Parameters: not used
08900      Other Information: returns device-specific dynamic
08950      status information in the word(s) immediately
09000      following the operation code
09050
09100

```

2) Line Frequency Clock

a) KWAIT

```

09300
09350      Class: Control
09400      Format restrictions: not applicable
09450      Byte Count: not used
09500      Buffer: not used
09550      Device Parameters: a one-word count (treated as an
09600      unsigned integer) denoting the number of 1/60
09650      second clock ticks ("jiffies") which are to elapse
09700      before a reply occurs.
09750

```

b) DSTATUS

```

09800
09850
09900      << not yet specified >>
09950

```

3) Line Printer

a) LPWRITE

```

10200      Class: Output
10250      Format restrictions: byte count is required
10300      Byte Count: must be even - rounded up if not
10350      Buffer: if last word is not full, high order (odd)
10400      byte should contain a pad of binary 0
10450      Device Parameters: none
10500

```

10550 The data contained in the buffer are transferred to the line
10600 printer, with a reply occurring upon completion of the transfer.
10650 The buffer should normally end with a line terminating character

10700 (e.g. line feed, form feed, vertical tab, carriage return,
10750 form feed, ↑S)
10800
10850 b) DSTATUS
10900
10950 << not yet specified>>
11000
11050 4) Teletype
11100
11150 a) TTREAD
11200
11250 Class: Input
11300 Format restrictions: none
11350 Byte Count: optional, as per format specification
11400 Buffer: required
11450 Device Parameters: none
11500
11550 When a complete line of input is available in the terminal'
**s
11600 input buffer, it will be copied into the user's buffer. A
11650 line is defined as a sequence of zero or more characters
11700 followed by a break character. Break characters are: line
11750 feed, carriage return, ↑A, ↑B, ↑C, ↑G, ↑K, ↑L, ↑Z, altmode. A t
**yped carriage
11800 return causes both the carriage return and a generated
11850 line feed to enter the buffer.
11900
11950 Rubout, ↑U, and limited type-ahead are handled by the
12000 teletype support in a manner analagous to the PDP-10. No
12050 break character definition, image mode, or
12100 full character set mode is available, nor will any of the
12150 above be provided until the terminal front-end system is
12200 completed. The existing teletype support is an interim
12250 stopgap package.
12300
12350 If the user-supplied buffer is inadequate to hold an
12400 entire typed line, as much of the line as will fit is supplied
12450 and a special reply is used (OPDONETYPE + LOSTINFOTYPE). The
12500 remainder of the input line is retained in the terminal's
12550 input buffer, and is supplied on the next input operation.
12600
12650 The terminal input buffer has a capacity of approximately
12700 120 characters.
12750
12800 If the user-supplied buffer resides within the i/o
12850 message itself, the size of the replied message can be used
12900 to determine the length of the line returned. If the buffer
12950 is specified indirectly, the break character which
13000 terminates the line is the only indication of line length.
13050
13100 b) TTWRITE
13150
13200 Class: Output
13250 Format restrictions: none

13300 Byte Count: optional, as per format specification
13350 Buffer: required
13400 Device Parameters: none
13450

If the byte count is omitted, the buffer is assumed to contain an ASCII string to be transmitted to the terminal. An explicit byte count causes the specified number of characters to be transmitted, including nulls if present. It is important to note that if an indirect buffer specification is used, the buffer must not be changed while the i/o request is in progress, since the output data is taken directly from the user's buffer. An attempt to do so will result in indeterminate output. This presents no restriction if the buffer is contained within the message itself, since the user will be unable to alter the message while the i/o system is processing it.

14100 c) DSTATUS

14150 << not yet specified >>
14200

14250 d) TTECHOCTL

14300 Class: control
14350 Format restrictions: not applicable
14400 Byte Count: not used
14450 Buffer: not used
14500 Device Parameters: The low order bit of the word
14550 following the operation code determines whether
14600 echoing is performed (bit = 1) or not (bit = 0).
14650
14700

14750 e) TTOUTRESET

14800 Class: control
14850 Format restrictions: not applicable
14900 Byte Count: not used
14950 Buffer: not used
15000 Device Parameters: none
15050
15100

15150 The successful execution of this operation causes all queued
15200 output requests, including the currently executing one, to
15250 be aborted (reply ERRTYPE). All program-generated output is
15300 thus canceled. Any pending echo characters are not affected
15350 by this operation.
15400

15450 f) TTINRESET

15500 Class: control
15550 Format restrictions: not applicable
15600 Byte Count: not used
15650 Buffer; not used
15700 Device Parameters: none
15750
15800

15850 The successful execution of this operation causes all pending
15900 input requests to be aborted (reply ERRTYPE). In addition,
15950

16000 if any complete or partial lines are present in the terminal
16050 input buffer, they are deleted. However, any pending echo
16100 for characters in the input buffer will be allowed to proceed.

16150
16200 g) TTINCLEAR

16250
16300 Class: control
16350 Format restrictions: not applicable
16400 Byte Count: not used
16450 Buffer: not used
16500 Device Parameters: none

16550
16600 This operation causes any pending input requests to be aborted
16650 (reply ERRTYPE). In addition, an implied ↑U is issued -- any
16700 partial line in the input buffer is deleted. Complete
16750 lines in the input buffer will be preserved, as will any pending
16800 echo characters.

16850
16900 h) TTEXCP

16950
17000 Class: control
17050 Format restrictions: not applicable
17100 Byte Count: not used
17150 Buffer: not used
17200 Device Parameters: not used

17250
17300 Only one TTEXCP request may be pending on a terminal at a
17350 time; any attempt to issue a second one will cause an
17400 immediate reply of ERRTYPE. TTEXCP remains pending until an
17450 unusual condition occurs, at which time a successful reply
17500 occurs and a word of information is returned in the location
17550 immediately following the TTEXCP opcode. If an unusual
17600 condition is detected when no TTEXCP request is pending, it
17650 is ignored. The conditions are:

17700
17750 TTSABREAK - break key was hit
17800 TTLOSTDATA - input rate too great
17850 TTSACTLO - ↑0 typed

17900
17950 5) DECTAPE

18000 a) TCSETUNIT

18100
18150 Class: Immediate
18200 Format restrictions: not applicable
18250 Byte Count: not used
18300 Buffer: not used
18350 Device Parameters: a unit number between 0 and 7
18400 inclusive in the word following the opcode.

18450
18500 If the specified unit number is available, it is allocated to
18550 the device, otherwise, the reply type REQILLDP is generated.
18600 When the DECTape connection is initially established (via
18650 PCONNECT), a unit number is allocated to it. Hence, TCSETUNIT

18700 need not be issued unless the initial unit number is unsatisfact
 18750 **ory.
 18800 This initial value may be determined by using the DIDENTIFY
 18850 operation.
 18900
 18950 b) TCREWIND
 19000 Class: control
 19050 Format restrictions: not applicable
 19100 Byte Count: not used
 19150 Buffer: not used
 19200 Device Parameters: none
 19250
 19300 The specified device is rewound to the forward end-zone, with
 19350 the reply being generated upon detection of the end-zone.
 19400
 19450 c) TCFINDBLOCK
 19500
 19550 Class: control
 19600 Format restrictions: not applicable
 19650 Byte Count: not used
 19700 Buffer: not used
 19750 Device Parameter: a one-word value specifying the
 19800 block at which the tape is to be positioned.
 19850
 19900 The tape is positioned so that an immediately following TCREAD
 19950 or TCWRITE specifying the same block number will experience
 20000 minimum positioning delay. If the block number cannot be
 20050 found on the tape, an error reply will occur (reply type
 20100 ERRTYPE).
 20150
 20200 d) TCREAD
 20250
 20300 Class: Input
 20350 Format restrictions: byte count required
 20400 Byte Count: should be even -- rounded up if not
 20450 Buffer: required
 20500 Device Parameter: a one-word value specifying the
 20550 block at which reading is to begin
 20600
 20650 If the specified block cannot be found, an error reply occurs
 20700 (ERRTYPE). Otherwise, input begins at the specified block and
 20750 continues (in a forward direction) until the count is exhausted.
 20800 Any "soft" error is retried five times before reporting the
 20850 failure.
 20900
 20950 e) TCWRITE
 21000
 21050 Class: Output
 21100 Format restrictions: byte count required
 21150 Byte Count: should be even -- rounded up if not
 21200 Buffer: required
 21250 Device Parameter: a one-word value specifying the
 21300 block at which writing is to begin

21350
21400
21450
21500
21550
21600
21650
21700

21750
21800
21850
21900
21950
22000
22050
22100
22150
22200
22250
22300
22350
22400
22450
22500
22550
22600
22650
22700
22750
22800
22850
22900
22950
23000
23050
23100
23150
23200
23250
23300
23350
23400
23450
23500
23550
23600
23650
23700
23750
23800
23850
23900
23950

Identical to TCREAD, but performs output instead of input.

DECTape errors:

Reply type ERRTYPE causes a single word of error information to be appended to i/o message. This type can be generated for TCREWIND, TCFINDBLOCK, TCREAD, and TCWRITE

**.

This word contains the value of the controller's status register (TCST) at the time the error occurred. Refer to Peripherals Manual for specific bit interpretations.

6) RP11 (moving head disk)

a) RPSEEK

Class: Control
Format restrictions: not applicable
Byte Count: none
Buffer: not used
Device Parameters: two words of disk addressing information, in a format described below

A seek operation is performed to position the read/write heads at a specified cylinder and track. No data transfer occurs. If the seek cannot be successfully performed, a reply with type ERRTYPE is generated, and error status information is returned in the message immediately following the device parameters.

b) RPREAD

Class: Input
Format restrictions: byte count required
Byte Count: should be even -- rounded up if not
Buffer: required
Device Parameters: two words of disk addressing information, in a format described below

The device parameters are used to seek the proper starting sector address. An input operation is then initiated which continues until the byte count has been exhausted. The transfer may involve more than one sector, and may cross track or cylinder boundaries. Error recovery is attempted, and "hard" errors are reported in the same way as for RPSEEK. See notes below for specifics.

c) RPWRITE

Class: Output
Format restrictions: byte count required
Byte Count: should be even -- rounded up if not
Buffer: required

24000
24050
24100
24150
24200
24250
24300
24350
24400
24450
24500
24550
24600
24650
24700
24750
24800
24850
24900
24950
25000
25050
25100
25150
25200
25250
25300
25350
25400
25450
25500
25550
25600
25650
25700
25750
25800
25850
25900
25950
26000
26050
26100
26150
26200
26250
26300
26350
26400
26450
26500
26550
26600
26650

Device Parameters: two words of disk addressing
information in a format described below.

Identical to RPREAD except that output is performed instead of
input.

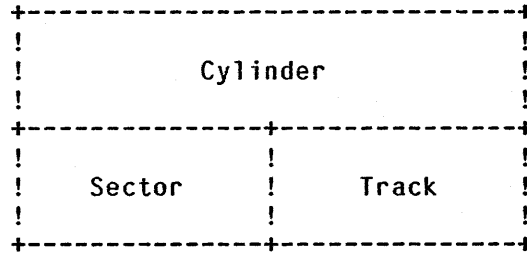
d) RPWRITECHECK

Class: Output
Format restrictions: byte count required
Byte Count: should be even -- rounded up if not
Buffer: required
Device Parameters: two words of disk addressing
information in a format described below

Identical to RPWRITE except that data from memory is compared
to data at the specified disk address. No data is actually
written on the disk. If a comparison error occurs, an error
reply (ERRTYPE) will occur, as described below.

Notes on RP11 i/o programming:

1. Device parameters take the following form:



Sector is not required for RPSEEK.

2. When an unrecoverable error occurs, the reply is of type
ERRTYPE, and two words of error status information are
returned in the message. The first of these is the
contents of RPER at the time of the error; the second
contains the value of RPDS. Refer to peripherals manual
for a description of the individual error bits.
3. Seek and transfer errors are retried five times by the
disk software before they are considered "hard" and
reported to the user program. Thus no further error
recovery need be attempted upon receipt of an ERRTYPE
reply.

7) ASLI Link (to another computer)

a) KLSETSPEED

Class: Control

26700 Format restrictions: not applicable
26750 Byte count: not used
26800 Buffer: not used
26850 Device parameters: one word containing line speed information

26900
26950 The parameter word contains a value in the range 0-7 in each of
27000 its bytes. The even byte specifies the line input speed; the odd byte
27050 specifies the output speed. The values have the following interpretations:

27100		
27150	0	110 Baud
27200	1	134.5
27250	2	300
27300	3	600
27350	4	1200
27400	5	2400
27450	6	4800
27500	7	9600
27550		

27600 The line is initialized to 4800 baud in, 300 baud out. These values
27650 are suitable for PDP-10 communication.

27700
27750 b) KLASCIIREAD

27800 Class: Input
27850 Format restrictions: byte count required
27900 Byte count: required
27950 Buffer: required
28000 Device parameters: none

28100 An input line of ASCII characters is assembled and placed in the
28150 buffer.
28200 If the buffer is of insufficient size to hold the entire line, the
28250 number of characters specified by the byte count is returned and LOSTIN
28300 FOTYPE is indicated with OPDONETYPE. No buffering is performed by the
28350 interrupt routine; hence, any characters which arrive when no i/o request
28400 is in effect will be discarded. The line break characters are the same
28450 as for TTREAD. If a hardware error is detected (break, lost data, etc.
28500), ERRTYPE will be indicated in the reply code and the value of the input status
28550 register will be returned in the word following the buffer.

28600
28650 c) KLBINARYREAD
28700

28750 Class: Input
 28800 Format restrictions: byte count required
 28850 Byte count: required
 28900 Buffer: required
 28950 Device parameters: none
 29000
 29050 Identical to KLASCIIREAD except that 8-bit characters are return
 **ed and
 29100 no break character processing is performed. Thus exhaustion of
 **the byte
 29150 count is the only terminating condition, and LOSTINFOTYPE is not
 ** indicated
 29200 with OPDONETYPE. A request specifying KLBINARYREAD will remain
 **pending
 29250 until the specified number of characters have been input.

d) KLWRITE

29450 Class: Output
 29500 Format restrictions: byte count required
 29550 Byte count: required
 29600 Buffer: required
 29650 Device parameters: none
 29700
 29750 Outputs the specified number of 8-bit characters. The character
 **s are
 29800 not interpreted in any way by the interrupt routine, so that any
 ** 8-bit
 29850 character is legal and will be transmitted unchanged.
 29900

.SUBSEC |Reply Codes|

30000 The i/o system generates a number of reply codes which describe
 30050 the outcome of a request. They are described in this section.
 30100
 30150
 30200 REQDEVDOWN - The device is no longer on-line.
 30250
 30300 REQTOOSMALL- The i/o request does not contain all of the
 30350 information required by the i/o system.
 30400
 30450 REQBADBUF - The buffer specification is illegal for one of
 30500 several reasons:
 30550 a) illegal cps slot
 30600 b) input operation and write-protected page
 30650 c) zero or negative byte count
 30700 d) buffer either crosses a page boundary
 30750 or is too large for message
 30800
 30850 REQILLFMT - Illegal format for specified opcode or unrecognized
 30900 REQILLOP opcode.
 30950
 31000 OPDONETYPE - Normal completion.
 31050
 31100 ERRRTYPE - Error completion.

31150
31200 In the event that completion (normal or error) occurs but not all
31250 of the desired information can be supplied (e.g. ERRTYPE return,
31300 but request is too small to hold error information), the value
31350 LOSTINFOTYPE is added to either ERRTYPE or OPDONETYPE to warn
31400 the program that not all the expected information is present.
31450
31500 .SUBSEC |Format Modifiers|
31550
31600 Two format modifiers are defined, INDBUF and NOCOUNT. INDBUF
31650 specifies that the buffer is addressed indirectly, as described earlier.
31700 NOCOUNT is used to indicate that the byte count has been omitted.
31750 These modifiers are ignored when used with operations which do not
31800 require a buffer.
31850
31900 To use a format modifier, the user program employs the
31950 IOOPN macro to define a composite operation code, e.g.
32000
32050 IOOPN(TTREAD,INDBUF+NOCOUNT)
↑L

00050 .SEC |THE APPENDIX|

00100
00150
00200
00250
00300
00350
00400

Except where necessary, absolute values and locations for fields are not given in this manual. The bindings for all symbolics may be found in the file HYKALL.R11[N810HY00] @ CMU-10A.

00450 .SUBSEC |HYDRA KERNEL RIGHTS|

00500
00550
00600
00650
00700
00750

In describing Hydra Kernel Rights, we consider the effect if Capability CAP has the right in question. If CAP is an Object Reference, we write OBJ as a shorthand for the Object Referenced by CAP:

- 00800 LOADRTS - Allows a Capability to be Loaded from OBJ
- 00850 STORTS - Allows a Capability to be Stored into OBJ
- 00900 APPRTS - Allows a Capability to be Appended onto OBJ
- 00950 KILLRTS - Allows a Capability to be Deleted from OBJ

01000
01050
01100
01150
01200

- 01050 GETRTS - Allows data to be gotten from OBJ
- 01100 PUTRTS - Allows data to be put into OBJ
- 01150 ADDRTS - Allows data to be appended onto OBJ

01250
01300
01350
01400
01450

- 01250 ALLYRTS - Allows OBJ to be Re-Allied
- 01300 OBJRTS - Allows OBJ to be Switced or Frozen
- 01400 CREARTS - Allows an Object to be Created from CAP
- 01450 COPYRTS - Allows a Copy to be made of OBJ

01500
01550
01600
01650
01700
01750
01800
01850
01900

- 01550 DLTRTS - Allows CAP to be Deleted
- 01600 ENVRTS - Allows CAP to be Stored in some Object
- 01650 MDFYRTS - Allows OBJ to be modified
- 01700 UCNFRTS - Allows OBJ to be Unconfined, that is, an Object accessed through OBJ may be modified.
- 01750 FRZRTS - Guarantees that OBJ is Frozen

01950 Note that the last set of 5 rights cannot be gained through rights amplification. Note that whenever rights are restricted, ALLYRTS are always removed as well.

02000
02050
02100

02150 .SUBSEC |RIGHTS RESTRICTION FORMAT|

02200
02250
02300
02350
02400
02450
02500

```

+---+---+---+---+---+---+---+---+---+---+
!                                     ! ! !                               !
02400 !           AUXRTS                !NF!TF!           UNUSED           !
02450 !           (8)                   ! ! !                (6)           !
+---+---+---+---+---+---+---+---+---+---+
02550 !                                     !
02600 !           KERNEL RIGHTS           !
02650 !           (1W)                    !

```



```

02700  +---'-----'-----'-----'+---'-----'-----'-----'+---
02750
02800      AUXRTS - Auxiliary rights
02850      NF - NEWFLAG
02900      TF - TEMPLFLAG
02950
03000      Calls that allow restriction of rights and flags (the flags fields, NF
03050      and TF are ignored in restricting an Object Reference) require an
03100      address that must point to a location in the active stack. That location
03150      is a two word area formatted as shown above. If the bit representing
03200      the particular Kernel or Auxiliary right or Flag is 0, the right or flag
03250      will be restricted.
03300
03350      Example, if the MUCH'th slot contained some Capability for a
03400      Procedure, to get a Capability for the same Procedure in the LESS'th
03450      slot having only CALLRTS, LNSRTS and DLTRTS, the following Bliss-11 code
03500      would do:
03550
03600          Begin
03650          Local RESTR[2];
03700          RESTR[0] ← CALLRTS or LNSRTS;
03750          RESTR[1] ← DLTRTS;
03800          Share ( LESS, MUCH, RESTR )
03850          End
03900
03950
04000  .SUBSEC |SIZE RESTRICTIONS|
04050
04100      The maximum size of a Data-Part is 1000 (#1750).
04150      The maximum number of Capabilities in a C-List is 125 (#175).
04200
04250
04300  .SUBSEC |KERNEL TYPES|
04350
04400
04450      For each Kernel Type, we specify a number of things:
04500
04550      a) Defined Auxiliary rights
04600      b) Initialization rights & flags - At system initialization, the
04650      initial Policy Subsystem has been provided with a Template with
04700      these rights and flags (NEWFLAG & TEMPLFLAG).
04750      c) Template rights and flags - The rights of a Template returned from
04800      the TEMPLATE Kall.
04850      d) Copy rights - The rights added when a Capability of that type
04900      is copied.
04950      e) Creation arguments - Additional arguments to the CREAT Kall.
05000      f) Copy arguments - Additional arguments to the COPY Kall.
05050
05100      - - - - -
05150
05200  1) Type TYPE
05250
05300      a) Auxiliary:
05350          TEMPLRTS - Allows Template of named Type to be made with all

```

```

05400             rights and flags.
05450             RTRVRTS - Allows TYPRETRIEVE Ka11
05500
05550 b) Initialization:
05600             LOADRTS, STORTS, APPRTS, KILLRTS, OBJRTS, CREARTS, COPYRTS,
05650             DLTRTS, ENVRTS, UCNFRTS, MDFYRTS, TEMPLFLAG, All Auxiliary rights
05700
05750 c) Template:
05800             DLTRTS, ENVRTS, TEMPLFLAG
05850
05900 d) Copy:
05950             DLTRTS
06000
06050 e) Creation arguments:
06100             Address (in stack) of 16 word area containing
06150             PNAME - words 1-5, Print Name
06200             CAPINIT - word 6, Initial C-List size of CREATED Object
06250             CAPMAX - word 7, Maximum C-List size
06300             DATAINIT - word 8, Initial Data-Part size
06350             DATAMAX - word 9, Maximum Data-Part size
06400             RTRVFLAG - word 10, Retrievability flag in sign bit.
06450
06500 f) Copy arguments:
06550             Same as Creation argument.
06600
06650 - - - - -
06700
06750 2) Type NULL
06800
06850 a) Auxiliary:
06900             NULLRTS - Determines whether Capability is Truenu11
06950
07000 b) Initialization:
07050             All Kernel and auxiliary rights, TEMPLFLAG. Note though that
07100             it is impossible to CREAT a Capability for a Null Object.
07150
07200 c) Template:
07250             All Kernel and Auxiliary rights, TEMPLFLAG.
07300
07350 d) Copy: May not be COPYed
07400
07450 e) Creation arguments: May not be CREATED
07500
07550 f) Copy arguments: May not be COPYed
07600
07650 - - - - -
07700
07750 3) Type PROCEDURE
07800
07850 a) Auxiliary:
07900             GETCBRTS - Allow access to ICB
07950             SETCBRTS - Allow modification of ICB
08000             PRCSRTS - Allows LNS incarnated from Procedure to initialize
08050             a Process

```

08100 LNSRTS - Allows LNS incarnated from Procedure to be LNSCALLED.
08150 CALLRTS - Allows Procedure to be CALLED.
08200
08250 b) Initialization:
08300 LOADRTS, STORTS, APPRTS, KILLRTS, OBJRTS, CREARTS, COPYRTS,
08350 DLTRTS, ENVRTS, UCNFRTS, MDFYRTS, TEMPLFLAG, All Auxiliary rights
08400
08450 c) Template:
08500 LOADRTS, STORTS, APPRTS, KILLRTS, OBJRTS, CREARTS, COPYRTS,
08550 DLTRTS, ENVRTS, MDFYRTS, TEMPLFLAG, All Auxiliary rights
08600
08650 d) Copy:
08700 DLTRTS
08750
08800 e) Creation arguments: None
08850
08900 f) Copy arguments: None
08950
09000 - - - - -
09050
09100 4) Type LNS
09150
09200 a) Auxiliary:
09250 GETCBRTS - Allows access to LCB
09300 SETCBRTS - Allows modification to LCB
09350 GSTKRTS - Allows access to LNS's active stack
09400 PSTKRTS - Allows modification of LNS's active stack
09450 PRCSRTS - Allows LNS to initialize a Process
09500 LNSRTS - Allows LNS to be LNSCALLED.
09550
09600 b) Initialization:
09650 DLTRTS, ENVRTS, TEMPLFLAG.
09700
09750 c) Template:
09800 DLTRTS, ENVRTS, TEMPLFLAG.
09850
09900 d) Copy: May not be COPYed
09950
10000 e) Creation arguments: May not be CREATED (See MAKLNS)
10050
10100 f) Copy arguments: May not be COPYed
10150
10200 Note: LNS Capabilities created with MAKLNS have the following rights:
10250 DLTRTS as well as UCNFRTS, FRZRTS, LNSRTS & PRCSRTS only if
10300 the Procedure it was incarnated from had those rights.
10350 LNS Capabilities created via the "Lns" argument specification
10400 for CALL have the following rights: LOADRTS, STORTS,
10450 APPRTS, KILLRTS, DLTRTS, MDFYRTS, GETCBRTS, SETCBRTS,
10500 GSTKRTS & PSTKRTS.
10550
10600 - - - - -
10650
10700 5) Type POLICY
10750

10800 a) Auxiliary:
10850 MAKERTS - Allows the MAKEPOLICY Ka11
10900 RCVRTS - Allows the RCVPOLICY Ka11
10950 POLRTS - Allows the POLICY Ka11
11000
11050 b) Initialization:
11100 LOADRTS, STORTS, APPRTS, KILLRTS, CREARTS, DLTRTS, ENVRTS,
11150 UCNFRTS, MDFYRTS, TEMPLFLAG, All Auxiliary rights
11200
11250 c) Template
11300 DLTRTS, ENVRTS, TEMPLFLAG
11350
11400 d) Copy: May not be COPYed
11450
11500 e) Creation arguments:
11550 One word indicating information about Policy Subsystem
11600 and its status
11650
11700 f) Copy arguments: May not be COPYed
11750
11800 - - - - -
11850
11900 6) Type PROCESS
11950
12000 a) Auxiliary:
12050 GETCBRTS - Allows access to PCB
12100 SETCBRTS - Allows modification to PCB
12150 STARTS - Allows the START Ka11
12200 STOPRTS - Allows the STOP Ka11
12250 CTLRTS - Allows the CONTROL Ka11
12300 SYNRTS - Allows the DESYNCH Ka11
12350 BASERTS - Allows association of Process Base in POLICY Ka11
12400 POLRTS - Allows association of Policy in POLICY Ka11
12450
12500 b) Initialization:
12550 CREARTS, DLTRTS, ENVRTS, UCNFRTS, MDFYRTS, TEMPLFLAG,
12600 All Auxiliary rights
12650
12700 c) Template:
12750 CREARTS, DLTRTS, ENVRTS, UCNFRTS, MDFYRTS, TEMPLFLAG,
12800 All Auxiliary rights except BASERTS
12850
12900 d) Copy: May not be COPYed
12950
13000 e) Creation arguments:
13050 Simple index denoting a Capability for an LNS Object
13100 with PRCSRTS. The LNS must be "useable" (See Subsection
13150 on PROCESS OBJECTS)
13200
13250 f) Copy arguments: May not be COPYed
13300
13350 - - - - -
13400
13450 7) Type PAGE

13500
13550
13600 a) Auxiliary:
13650 CPSRTS - Allows Page to be loaded into CPS
13700 PGWRTS - Allows Page to be written into
13750 CACHRTS - Allows Page to be cached
13800
13850 b) Initialization:
13900 OBJRTS, CREARTS, COPYRTS, DLTRTS, ENVRTS, UCNFRTS, MDFYRTS,
13950 TEMPLFLAG, All Auxiliary rights
14000
14050 c) Template:
14100 OBJRTS, CREARTS, COPYRTS, DLTRTS, ENVRTS, UCNFRTS, MDFYRTS,
14150 TEMPLFLAG, CPSRTS, PGWRTS
14200
14250 d) Copy:
14300 OBJRTS, DLTRTS, ENVRTS, UCNFRTS, MDFYRTS, CPSRTS, PGWRTS
14350
14400 e) Creation arguments: None
14450
14500 f) Copy arguments:
14550 Index of a CPS slot. The COPYed PAGE will be CPSLOADed into
14600 that CPS slot.
14650 - - - - -
14700
14750 8) Type SEMAPHORE
14800
14850 a) Auxiliary: None
14900
14950 b) Initialization:
15000 CREARTS, DLTRTS, ENVRTS, UCNFRTS, MDFYRTS, TEMPLFLAG
15050
15100 c) Template:
15150 DLTRTS, ENVRTS, TEMPLFLAG
15200
15250 d) Copy: May not be COPYed
15300
15350 e) Creation arguments:
15400 Initial value of Semaphore
15450
15500 f) Copy arguments: May not be COPYed
15550 - - - - -
15600
15650
15700 9) Type POLSEM
15750
15800 a) Auxiliary:
15850 PRTS - Allows the PPOLSEM Kall
15900 VRTS - Allows the VPOLSEM Kall
15950 CPRTS - Allows the CPOLSEM Kall
16000
16050 b) Initialization:
16100 CREARTS, DLTRTS, ENVRTS, UCNFRTS, MDFYRTS, TEMPLFLAG,
16150 All Auxiliary rights

16200
16250 c) Template:
16300 DLTRTS, ENVRTS, TEMPLFLAG
16350
16400 d) Copy: May not be COPYed
16450
16500 e) Creation arguments:
16550 Initial value of the Policy Semaphore
16600
16650 f) Copy arguments: May not be COPYed
16700
16750 - - - - -
16800
16850 10) Type DATA
16900
16950 a) Auxiliary: None
17000
17050 b) Initialization:
17100 GETRTS, PUTRTS, ADDRRTS, OBJRTS, CREATS, COPYRTS, DLTRTS,
17150 ENVRTS, UCNFRTS, MDFYRTS, TEMPLFLAG
17200
17250 c) Template:
17300 GETRTS, PUTRTS, ADDRRTS, OBJRTS, CREATS, COPYRTS, DLTRTS,
17350 ENVRTS, UCNFRTS, MDFYRTS, TEMPLFLAG
17400
17450 d) Copy:
17500 GETRTS, PUTRTS, ADDRRTS, OBJRTS, DLTRTS, ENVRTS, UCNFRTS, MDFYRTS
17550
17600 e) Creation arguments: None
17650
17700 f) Copy arguments:
17750 Length of Data-Part of COPYed Object. The Data-Part of the
17800 COPYed Object will be expanded or contracted as necessary. If
17850 less than or equal to 0, the length will be as in the original.
17900
17950 - - - - -
18000
18050 11) Type UNIVERSAL
18100
18150 a) Auxiliary: None
18200
18250 b) Initialization:
18300 LOADRTS, STORTS, APPRTS, KILLRTS, GETRTS, PUTRTS, ADDRRTS,
18350 OBJRTS, CREATS, COPYRTS, DLTRTS, ENVRTS, UCNFRTS, MDFYRTS,
18400 TEMPLFLAG
18450
18500 c) Template:
18550 LOADRTS, STORTS, APPRTS, KILLRTS, GETRTS, PUTRTS, ADDRRTS,
18600 OBJRTS, CREATS, COPYRTS, DLTRTS, ENVRTS, UCNFRTS, MDFYRTS,
18650 TEMPLFLAG
18700
18750 d) Copy:
18800 LOADRTS, STORTS, APPRTS, KILLRTS, GETRTS, PUTRTS, ADDRRTS,
18850 OBJRTS, DLTRTS, ENVRTS, UCNFRTS, MDFYRTS

18900
 18950 e) Creation arguments: None
 19000
 19050 f) Copy arguments:
 19100 Same as for DATA.
 19150
 19200 - - - - -
 19250
 19300 12) Type PORT
 19350
 19400 a) Auxiliary:
 19450 PCONNRTS - Allows PCONNECT and PDISCONNECT Calls
 19500 MCREARTS - Allows MCREATE Call
 19550 MWRTIRTS - Allows MWRITE Call
 19600 MREADRTS - Allows MREAD Call
 19650 MSENDRTS - Allows MSEND Call
 19700 MRSVPRTS - Allows MRSVP Call
 19750 MRPLYRTS - Allows MREPLY Call
 19800 MWAIRTS - Allows MWAIT Call
 19850
 19900 b) Initialization:
 19950 CREARTS, DLTRTS, ENVRTS, UCNFRTS, MDFYRTS, TEMPLFLAG,
 20000 All Auxiliary rights
 20050
 20100 c) Template:
 20150 CREARTS, DLTRTS, ENVRTS, UCNFRTS, MDFYRTS, TEMPLFLAG,
 20200 All Auxiliary rights
 20250
 20300 d) Copy: May not be COPYed
 20350
 20400 e) Creation arguments:
 20450 To be specified
 20500
 20550 f) Copy arguments: May not be COPYed
 20600
 20650 - - - - -
 20700
 20750 13) Type DEVICE
 20800
 20850 a) Auxiliary:
 20900 PCONNRTS - Allows PCONNECT and PDISCONNECT Calls
 20950 Rest to be specified
 21000
 21050 b) Initialization:
 21100 CREARTS, DLTRTS, ENVRTS, UCNFRTS, MDFYRTS, TEMPLFLAG,
 21150 All Auxiliary rights
 21200
 21250 c) Template:
 21300 DLTRTS, ENVRTS, TEMPLFLAG
 21350
 21400 d) Copy:
 21450 DLTRTS, Rest to be specified
 21500
 21550 e) Creation arguments:

21600 To be specified
21650
21700 f) Copy arguments:
21750 To be specified
21800
21850 - - - - -
21900
21950
22000 †L

22050 .SUBSEC |FORMAT FOR WHAT|

22100

22150 The WHAT Kall provides a representation of a Capability. The format