NETWORK INSTALLATION GUIDE

for

CTI-NET

Version 1.0

Implementation Notes:

1> A newer version of 'ps' is include in the 'bin' directory, this may be needed if you are running a version of the 'idris' resident prior to 3.14.

2> The current version of the standard header file 'stat.h' is out of date, a later version is in '/usr/local/bin/stat.h'. The network include file 'include/sys/stat.h' has a temporary modification to use this later version. It should be changed back when the header file 'lib/hdrs/stat.h' is updated.

3> To compile the network code, most parts of the 'C' compiler must have a higher value of 'setb'. Also 'db' is needed for the 'MAKEBIG' script to run.

4> The following changes where made to the resident directory:

main.c	- added to the CDEVSW and nibble table, added include for 'net_space.h'
sio582.c	- added modifications for 'select' and 'slip'.
upd7201.c	- added modifications for 'select' and 'slip'.
a.s	- added call to 'net_init()', changed interrupt mask, added extra vector entried.
Makefile	- fixed to compile new files, fixed for v315.
c68k.pro	- added new include paths, added defines for 'pp'.
/idris	- must keep symbol-table for 'netstat' to work.
debug.c	- new file needed for v315 resident.
spl.c	- new file needed for v315 resident.
et3c400.c	- new file, ethernet device driver.
MAKEBIG	- shell script run on 'resident' after 'mk' to increase internal structure sizes for network.
network.h	- new file to enable network or not.
netlib.d	- new file, kernel network functions.
buserr.c	- newer version for v315 resident.
*.d	- newer version for v315 resident.

5> Remote logins (telnet) do no inherit the environment set up in '/adm/init' for security reasons. Because of this, the files '$HOME/.login' and '/adm/.login' must set up any needed environment.

6> Since the log file for 'inetd' grows everytime a remote login or remote 'ftp' is serviced, it should be cleaned every now and then, or the log file can be set to '/dev/null'.

DISK-1 FILES:

```
-rwxrwxrwx   0   0     217091 Dec 02 21:15 idris.net
drwxrwxrwx   0   0          0 Dec 02 20:48 templates/
-rw-r--r--   0   0        405 Dec 02 20:49 templates/hosts
-rw-r--r--   0   0        418 Dec 02 20:49 templates/inetd.conf
-rw-r--r--   0   0        849 Dec 02 20:48 templates/init
-rw-r--r--   0   0         83 Dec 02 20:49 templates/login.adm
-rw-r--r--   0   0        171 Dec 02 20:49 templates/login.root
-rw-r--r--   0   0         27 Dec 02 20:49 templates/logout.adm
-rw-r--r--   0   0        190 Dec 02 20:49 templates/networks
-rw-r--r--   0   0        170 Dec 02 20:49 templates/protocols
-rw-r--r--   0   0        230 Dec 02 20:49 templates/services
drwxrwxrwx   0   0          0 Dec 02 20:32 lib/
-rwx------   0   0      28990 Dec 02 20:51 lib/dstaddr
-rwx------   0   0      74735 Dec 02 20:51 lib/ftpd
-rwx------   0   0      34692 Dec 02 20:51 lib/ifconfig
-rwx------   0   0      29510 Dec 02 20:51 lib/inetd
-rw-r--r--   0   0      12957 Dec 02 20:52 lib/libnet.o
-rwx------   0   0      37149 Dec 02 20:51 lib/route
-rwx------   0   0      18180 Dec 02 20:51 lib/slattach
-rwx------   0   0      40199 Dec 02 20:51 lib/telnetd
drwxrwxrwx   0   0          0 Dec 02 20:34 bin/
-rwxr-xr-x   0   0      49677 Dec 02 20:53 bin/db
-rwxr-xr-x   0   0      56474 Dec 02 20:53 bin/ftp
-rwxr-xr-x   0   0      17946 Dec 02 20:53 bin/hostid
-rwxr-xr-x   0   0      17257 Dec 02 20:53 bin/hostname
-rwxr-xr-x   0   0      65573 Dec 02 20:53 bin/netstat
-rwxr-xr-x   0   0      37458 Dec 02 20:53 bin/ping
-rwxr-xr-x   0   0      28213 Dec 02 20:52 bin/ps.314
-rw-r--r--   0   0       2622 Dec 02 20:53 bin/stat.h
-rwxr-xr-x   0   0      53868 Dec 02 20:53 bin/telnet
30 files tabulated
```

DISK-2 FILES:

```
drwxrwxrwx   0   0         0 Dec 02 20:23 hdrs/
-rw-r--r--   0   0       344 Dec 02 20:54 hdrs/alloc.h
-rw-r--r--   0   0       898 Dec 02 20:54 hdrs/bio.h
-rw-r--r--   0   0      1079 Dec 02 20:54 hdrs/cio.h
-rw-r--r--   0   0      1863 Dec 02 20:54 hdrs/cpu.h
-rw-r--r--   0   0       153 Dec 02 20:54 hdrs/dir.h
-rw-r--r--   0   0       660 Dec 02 20:54 hdrs/errno.h
-rw-r--r--   0   0       774 Dec 02 20:54 hdrs/fcntl.h
-rw-r--r--   0   0      2766 Dec 02 20:54 hdrs/fio.h
-rw-r--r--   0   0       126 Dec 02 20:54 hdrs/grp.h
-rw-r--r--   0   0       777 Dec 02 20:54 hdrs/ino.h
-rw-r--r--   0   0       684 Dec 02 20:54 hdrs/limits.h
-rw-r--r--   0   0        59 Dec 02 20:54 hdrs/mi68k.h
-rw-r--r--   0   0      2085 Dec 02 20:54 hdrs/msg.h
-rw-r--r--   0   0      2436 Dec 02 20:54 hdrs/msys.h
-rw-r--r--   0   0       337 Dec 02 20:54 hdrs/pan68k.h
-rw-r--r--   0   0       171 Dec 02 20:54 hdrs/pwd.h
-rw-r--r--   0   0      2959 Dec 02 20:54 hdrs/res.h
-rw-r--r--   0   0      1735 Dec 02 20:54 hdrs/rvax.h
-rw-r--r--   0   0        70 Dec 02 20:54 hdrs/setjmp.h
-rw-r--r--   0   0      2226 Dec 02 20:54 hdrs/stat.h
-rw-r--r--   0   0      5038 Dec 02 20:54 hdrs/std.h
-rw-r--r--   0   0       816 Dec 02 20:54 hdrs/sup.h
-rw-r--r--   0   0       408 Dec 02 20:54 hdrs/swp.h
drwxrwxrwx   0   0         0 Dec 02 20:23 hdrs/sys/
-rw-r--r--   0   0       841 Dec 02 20:54 hdrs/sys/badblk.h
-rw-r--r--   0   0      1675 Dec 02 20:54 hdrs/sys/idris.h
-rw-r--r--   0   0       383 Dec 02 20:54 hdrs/sys/signal.h
-rw-r--r--   0   0      1991 Dec 02 20:54 hdrs/sys/stat.h
-rw-r--r--   0   0       183 Dec 02 20:54 hdrs/sys/times.h
-rw-r--r--   0   0       206 Dec 02 20:54 hdrs/sys/types.h
-rw-r--r--   0   0       246 Dec 02 20:54 hdrs/sys/utsname.h
-rw-r--r--   0   0      4098 Dec 02 20:54 hdrs/termio.h
-rw-r--r--   0   0       686 Dec 02 20:54 hdrs/time.h
-rw-r--r--   0   0      1304 Dec 02 20:54 hdrs/unistd.h
-rw-r--r--   0   0      1308 Dec 02 20:54 hdrs/usr.h
-rw-r--r--   0   0       765 Dec 02 20:54 hdrs/ustat.h
-rw-r--r--   0   0       161 Dec 02 20:54 hdrs/utime.h
-rw-r--r--   0   0       196 Dec 02 20:54 hdrs/xecv.h
-rw-r--r--   0   0       535 Dec 02 20:54 hdrs/xeq.h
drwxrwxrwx   0   0         0 Dec 02 20:24 include/
drwxrwxrwx   0   0         0 Dec 02 20:23 include/arpa/
-rw-r--r--   0   0      2242 Dec 02 20:54 include/arpa/ftp.h
-rw-r--r--   0   0      3888 Dec 02 20:54 include/arpa/telnet.h
```

DISK-2 FILES: (continued) (page-2)

```
drwxrwxrwx   0   0        0 Dec 02 20:23 include/net/
-rw-r--r--   0   0     1113 Dec 02 20:55 include/net/af.h
-rw-r--r--   0   0     7556 Dec 02 20:55 include/net/if.h
-rw-r--r--   0   0     2176 Dec 02 20:55 include/net/if_arp.h
-rw-r--r--   0   0     1557 Dec 02 20:55 include/net/netisr.h
-rw-r--r--   0   0     1886 Dec 02 20:55 include/net/raw_cb.h
-rw-r--r--   0   0     2713 Dec 02 20:55 include/net/route.h
-rw-r--r--   0   0     2348 Dec 02 20:55 include/netdb.h
drwxrwxrwx   0   0        0 Dec 02 20:23 include/netimp/
-rw-r--r--   0   0     7968 Dec 02 20:55 include/netimp/if_imp.h
-rw-r--r--   0   0     3541 Dec 02 20:55 include/netimp/if_imphost.h
drwxrwxrwx   0   0        0 Dec 02 20:23 include/netinet/
-rw-r--r--   0   0     1301 Dec 02 20:55 include/netinet/icmp_var.h
-rw-r--r--   0   0     2610 Dec 02 20:55 include/netinet/if_ether.h
-rw-r--r--   0   0     3170 Dec 02 20:55 include/netinet/in.h
-rw-r--r--   0   0     1579 Dec 02 20:55 include/netinet/in_pcb.h
-rw-r--r--   0   0     1149 Dec 02 20:55 include/netinet/in_systm.h
-rw-r--r--   0   0     1781 Dec 02 20:55 include/netinet/in_var.h
-rw-r--r--   0   0     5511 Dec 02 20:55 include/netinet/ip.h
-rw-r--r--   0   0     3995 Dec 02 20:55 include/netinet/ip_icmp.h
-rw-r--r--   0   0     3525 Dec 02 20:55 include/netinet/ip_var.h
-rw-r--r--   0   0     2453 Dec 02 20:55 include/netinet/tcp.h
-rw-r--r--   0   0      995 Dec 02 20:55 include/netinet/tcp_debug.h
-rw-r--r--   0   0     2273 Dec 02 20:55 include/netinet/tcp_fsm.h
-rw-r--r--   0   0     1334 Dec 02 20:55 include/netinet/tcp_seq.h
-rw-r--r--   0   0     4473 Dec 02 20:55 include/netinet/tcp_timer.h
-rw-r--r--   0   0     8533 Dec 02 20:55 include/netinet/tcp_var.h
-rw-r--r--   0   0     1202 Dec 02 20:55 include/netinet/tcpip.h
-rw-r--r--   0   0      791 Dec 02 20:55 include/netinet/udp.h
-rw-r--r--   0   0     1243 Dec 02 20:55 include/netinet/udp_var.h
drwxrwxrwx   0   0        0 Dec 02 20:24 include/netns/
-rw-r--r--   0   0      971 Dec 02 20:56 include/netns/idp.h
-rw-r--r--   0   0      922 Dec 02 20:56 include/netns/idp_var.h
-rw-r--r--   0   0     3111 Dec 02 20:56 include/netns/ns.h
-rw-r--r--   0   0     2581 Dec 02 20:56 include/netns/ns_error.h
-rw-r--r--   0   0     1832 Dec 02 20:56 include/netns/ns_if.h
-rw-r--r--   0   0     1875 Dec 02 20:56 include/netns/ns_pcb.h
-rw-r--r--   0   0     1155 Dec 02 20:56 include/netns/sp.h
-rw-r--r--   0   0     1233 Dec 02 20:56 include/netns/spidp.h
-rw-r--r--   0   0      996 Dec 02 20:56 include/netns/spp_debug.h
-rw-r--r--   0   0     3958 Dec 02 20:56 include/netns/spp_timer.h
-rw-r--r--   0   0     7524 Dec 02 20:56 include/netns/spp_var.h
-rw-r--r--   0   0     1964 Dec 02 20:55 include/nlist.h
-rw-r--r--   0   0     1765 Dec 02 20:55 include/strings.h
```

DISK-2 FILES: (continued) (page-3)

```
drwxrwxrwx   0   0        0 Dec 02 20:24 include/sys/
-rw-r--r--   0   0     1674 Dec 02 20:56 include/sys/buf.h
-rw-r--r--   0   0     1680 Dec 02 20:56 include/sys/ctype.h
-rw-r--r--   0   0     2075 Dec 02 20:56 include/sys/dir.h
-rw-r--r--   0   0     1671 Dec 02 20:56 include/sys/dk.h
-rw-r--r--   0   0     1139 Dec 02 20:56 include/sys/domain.h
-rw-r--r--   0   0     5338 Dec 02 20:56 include/sys/errno.h
-rw-r--r--   0   0     1737 Dec 02 20:56 include/sys/ether.h
-rw-r--r--   0   0     1787 Dec 02 20:56 include/sys/fcntl.h
-rw-r--r--   0   0     1801 Dec 02 20:56 include/sys/file.h
-rw-r--r--   0   0     1671 Dec 02 20:56 include/sys/hy.h
-rw-r--r--   0   0     1674 Dec 02 20:56 include/sys/imp.h
-rw-r--r--   0   0     2236 Dec 02 20:56 include/sys/inode.h
-rw-r--r--   0   0     7737 Dec 02 20:56 include/sys/ioctl.h
-rw-r--r--   0   0     1815 Dec 02 20:56 include/sys/kernel.h
-rw-r--r--   0   0     5647 Dec 02 20:56 include/sys/mbuf.h
-rw-r--r--   0   0     3539 Dec 02 20:56 include/sys/param.h
-rw-r--r--   0   0     2690 Dec 02 20:56 include/sys/proc.h
-rw-r--r--   0   0     7558 Dec 02 20:56 include/sys/protosw.h
-rw-r--r--   0   0     2466 Dec 02 20:56 include/sys/sgtty.h
-rw-r--r--   0   0     2382 Dec 02 20:56 include/sys/signal.h
-rw-r--r--   0   0     1671 Dec 02 20:56 include/sys/sl.h
-rw-r--r--   0   0     4598 Dec 02 20:56 include/sys/socket.h
-rw-r--r--   0   0     5562 Dec 02 20:56 include/sys/socketvar.h
-rw-r--r--   0   0     2066 Dec 02 20:56 include/sys/stat.h
-rw-r--r--   0   0     1772 Dec 02 20:56 include/sys/syslog.h
-rw-r--r--   0   0     1728 Dec 02 20:56 include/sys/systm.h
-rw-r--r--   0   0     1858 Dec 02 20:56 include/sys/time.h
-rw-r--r--   0   0     2258 Dec 02 20:56 include/sys/tty.h
-rw-r--r--   0   0     2156 Dec 02 20:56 include/sys/types.h
-rw-r--r--   0   0     2130 Dec 02 20:56 include/sys/uio.h
-rw-r--r--   0   0      906 Dec 02 20:56 include/sys/un.h
-rw-r--r--   0   0     2348 Dec 02 20:56 include/sys/unpcb.h
-rw-r--r--   0   0     2690 Dec 02 20:56 include/sys/user.h
drwxrwxrwx   0   0        0 Dec 02 21:33 res.dmi/
-rwxrwxrwx   0   0    12110 Dec 02 20:24 res.dmi/68kres.d
-rw-rw-rw-   0   0      445 Dec 02 20:24 res.dmi/MAKEBIG
-rw-rw-r--  44  31     1247 Dec 02 20:27 res.dmi/Makefile
-rw-rw-r--  44  31      784 Dec 02 20:24 res.dmi/Makefloppy
-rw-rw-r--  44  31     6961 Dec 02 20:24 res.dmi/a.s
-rw-rw-r--  44  31      601 Dec 02 20:24 res.dmi/addsys.s
-rw-rw-rw-   0   0     2196 Dec 02 20:24 res.dmi/buserr.c
-rw-r--r--   0   0     2713 Dec 02 20:24 res.dmi/c68k.pro
-rw-rw-rw-   0   0     5965 Dec 02 20:24 res.dmi/debug.c
-rw-rw-rw-   0   0      134 Dec 02 20:24 res.dmi/debug.h
-rw-rw-rw-   0   0    20986 Dec 02 20:24 res.dmi/et3c400.c
```

DISK-2 FILES: (continued)

```
-rw-rw-r--  44  31      8620 Dec 02 20:24 res.dmi/fdc235.c
-rw-rw-r--  44  31      8452 Dec 02 20:24 res.dmi/fdmain.c
-rw-rw-r--  44  31     10701 Dec 02 20:24 res.dmi/main.c
-rw-rw-r--  44  31       190 Dec 02 20:24 res.dmi/makedate.h
-rw-rw-rw-   0   0      3164 Dec 02 20:25 res.dmi/mc68451.d
-rw-rw-r--  44  31      5260 Dec 02 20:25 res.dmi/md.c
-rw-rw-r--  44  31      2786 Dec 02 20:25 res.dmi/mf.c
-rwxrwxrwx   0   0     10088 Dec 02 20:25 res.dmi/msys.d
-rw-rw-rw-   0   0      4519 Dec 02 20:25 res.dmi/net_space.h
-rwxrwxrwx   0   0     97341 Dec 02 20:25 res.dmi/netlib.d
-rw-rw-rw-   0   0      1775 Dec 02 20:25 res.dmi/network.h
-rw-rw-rw-   0   0       892 Dec 02 20:25 res.dmi/nmi.c
-rw-rw-r--  44  31      8965 Dec 02 20:25 res.dmi/sio582.c
-rw-rw-r--   0   0       683 Dec 02 20:25 res.dmi/spl.s
-rwxrwxrwx   0   0     61016 Dec 02 20:25 res.dmi/srcres.d
-rw-rw-r--  44  31      9376 Dec 02 20:25 res.dmi/upd7201a.c
-rw-rw-r--  44  31      5813 Dec 02 20:25 res.dmi/wdc235.c
drwxrwxrwx   0   0         0 Dec 02 21:01 boot.dmi/
-rw-rw-r--   0   0       698 Dec 02 20:24 boot.dmi/Makefile
-rw-rw-r--   0   0      1293 Dec 02 20:58 boot.dmi/c68k.pro
-rw-rw-r--   0   0      9908 Dec 02 20:24 boot.dmi/cboot.c
-rw-rw-r--   0   0       182 Dec 02 20:24 boot.dmi/conf.c
-rw-rw-r--   0   0       360 Dec 02 20:24 boot.dmi/dschdr.s
-rw-rw-r--   0   0      3425 Dec 02 20:24 boot.dmi/fdc.c
-rw-rw-r--   0   0       725 Dec 02 20:24 boot.dmi/io7201.c
-rw-rw-r--   0   0      2570 Dec 02 20:24 boot.dmi/wdc.c
160 files tabulated
```

Installation

With the addition of the networking code, the 'idris' kernel is too large to be loaded with the current 'cboot' program. The 'cboot' program needs to be re-linked and re-installed on the hard disk, before changing the kernel. What needs to be done is to change the 'Makefile' to link 'cboot' at 0x40000 instead of 0x20000. This will cause 'cboot' to get loaded higher in memory, so that there is enough room to load the 'idris' kernel below it. After the new 'cboot' program has been re-made, place it on the hard disk using the command 'dd -o /dev/wd0 cboot'. The system should be re-booted to insure that the current kernel can still be loaded with the new 'cboot' program.

Directories for the 'user' and 'system' executable programs must be created. These 'user' files might be placed in the normal '/bin' directory, a better location might be '/usr/local/bin'. The 'system' executables might be placed in the '/odd' directory, but again a better location might be in '/usr/local/lib'. Available disk space and disk partitioning will probably be factors in the final selections. The location of the 'user' files should be added to the 'PATH' variables, as set in the various user's '.login' files.

User Files	System Files
ping	inetd
telnet	telnetd
ftp	ftpd
hostname	route
hostid	ifconfig
netstat	slattach
	dstaddr

Install the new 'idris' kernel file, it must either be installed as '/idris' or have a link by that name. This is required for proper operation of the 'netstat' program. The '/idris' file must still have its symbol-table, 'netstat' uses it to locate kernel variables in '/dev/kmem'. Be sure to keep a copy of the old kernel file around until you are sure the new one works.

Create the needed devices in the '/dev' directory. To do this, first create two new directories '/dev/socket' and '/dev/select'. The easiest way to build the new devices is to re-boot using the new '/idris' kernel, stay single user, and then:

```
cd /dev
mkdev -c0 -i < /dev/cnames
mkdir /dev/socket /dev/select
mv sock?? socket    (warning that can't move 'socket' is ok)
mv sel??  select
```

Besides creating the 'socket' and 'select' devices, this will also create the 'ptyp?' and 'ttyp?' devices. They are already in the correct directory and do not need to be moved.

Copy the following files to the '/adm' directory:

```
hosts
networks
protocols
services
inetd.conf
ftpusers    (optional)
```

Edit the 'hosts' and 'networks' files to reflect your network configuration. For every host that you will have on the network, add an entry to the '/adm/host' file. Each network should have an entry in the 'adm/networks' file. The system will operate without these files, but without them you can only refer to hosts by their Internet addresses, not by an easy to remember name. The '/adm/networks' file is mainly used by 'netstat', for display purposes. The files 'protocols', 'services', and 'inetd.conf' don't require any changes, just use them as provided.

In order to use the 'ftp' program you, must have passwords on the user names that you will be logging in with. If desired, add an entry to '/adm/passwd' for 'anonymous' 'ftp' access. The entry for 'anonymous' 'ftp' access is a standard 'passwd' entry, with the user name of 'ftp' and any password. It must have a password, but it is not used for this special 'ftp' access. This special entry allows an 'ftp' login with the user names 'anonymous' or 'ftp'. A password will still be asked for, but anything will be accepted. The user will only be allowed access to the home directory, as specified for the 'ftp' entry in the 'passwd' file. A 'chroot' call is made to this directory. This special access can be used to allow limited public access to the system, if desired.

If desired, add entries to '/adm/ftpusers'. This file is used to disallow 'ftp' access for specific user names. Prohibited user names should each appear on a separate line with nothing else, including spaces, before or after the name.

Add the network startup commands to '/adm/init'. The following example assumes that your local hostname is 'miny' and that the standard file placement was used. These lines should be placed after the single-user shell, after the date is set, and after any commands to mount other disk partitons. Just prior to going 'multi-user' is a good place. All the comments (lines beginning with '#') may be deleted if desired.

Example '/adm/init' lines:

```
(start this right after the single user shell)
#
W echo "Starting Network"
# set the local hostname to 'miny'
W /usr/local/bin/hostname miny
#
# the 'hostid' command is optional, it is not needed.
W /usr/local/bin/hostid 123456789
#
# these three commands are needed to start a 'slip' connection
# the name of the machine at the remote end is 'mega'.
# if there is more than one 'ifconfig', the addresses must be
# different, the address are either  specified as numeric
# addresses or names from the '/adm/host' files.
# 'slattach' must be run in the background to keep port open.
S /usr/local/lib/slattach /dev/tty0 9600
W /usr/local/lib/dstaddr  sl0 mega
W /usr/local/lib/ifconfig sl0 sminy
#
# this is all that is needed to start an ethernet interface.
# the '-trailers' makes it more compatible with other hosts.
# a driver message will be sent to the console showing the
# board's memory address and its ehternet address.
W /usr/local/lib/ifconfig et0 miny -trailers
#
# this starts the server for remote services, runs in background.
S /usr/local/lib/inetd /adm/inetd.conf /adm/inetd.log /usr/local/lib
#
W echo "Network Started"
#
(just prior to '/bin/echo "multi user"')
```

Once all the above is complete, you should reboot the system before going multi-user. Be sure to use 'sync' a couple of times before you reboot. The network will not be running until the machine is in the 'multi-user' mode.

Here is an example of a user '.login' file:

```
    /usr/local/bin/hostname | set P -i
    /bin/printenv LOGNAME   | set Z -i
    set P ($Z@$P):" "
    set Z
    set X '|/usr/local/bin/|/etc/$X'
    setenv CPATH /
    setenv SHELL /bin/sh
    mail -q
```

FTP

ftp [-vnigs] [-d#] -[f[filename]] [host]

The 'ftp' program is used to transfer files between hosts. The three leters in 'ftp' stand for '**F**ile **T**ransfer **P**rotocol'.

Flags:

v - turns on the verbose mode, shows server responses, the verbose mode is the default so really of little use.

n - normally when ftp makes a connection it starts the login sequence by asking for the 'user' name, this flags turns this off, instead ftp will just enter the command mode.

i - the default for ftp is to prompt for each file in the multiple commands, this turns the interactive mode off.

g - normally ftp starts with file globbing enabled, this flag allows ftp to start with it off.

s - normally ftp converts '\' to '/'. This allows files to be specified in either the 'unix' or 'dos' format. This flags disables the '\' translation.

d - this flag allows a debug level to be specified in the range of 0-9. 0 turns off debug, 1 gives the lowest level of debug, and 9 gives the most.

f - this flag allows a filename to be specified from which ftp will read it input, instead of the console.

The optional 'host' parameter specifies an initial host to try to connect to. It it is not specified, then 'ftp' enters the command mode.

Commands:

?	- print the local help information.
!	- escape to a local interactive shell, or if a command is given, pass it to a local shell then return to command mode when it is done.
ascii	- set the default file transfer type to 'ascii'.
bell	- ring the console bell whenever a command is completed (toggle).

FTP (continued) (page-2)

Commands: (continued)

bget	- get a file, force the use of the 'binary' file transfer type.
binary	- set the default file transfer type to 'binary'.
bput	- put a file, force the use of the 'binary' file transfer type.
bye	- terminate the 'ftp' session and exit the 'ftp' program.
cd	- change the remote working directory.
close	- terminate the 'ftp' session, but stay in the 'ftp' command mode.
delete	- delete a remote file - inquires if prompting is on.
debug	- toggle/set the debugging level.
dir	- do a 'list' command to get the directory listing from the remote host.
get	- get a file using the default file transfer type.
glob	- toggle the metacharacter expansion of local file names.
hash	- toggle the printing of a '#' character for each buffer transferred.
help	- print the local help information. If no argument then prints a list of the valid commands. If a command is specified then a short description of it will be given.
interactive	- turn on the prompting for multiple commands.
lcd	- change the local working directory.
lls	- list the contents of the local directroy.
ls	- do a 'nlist' command to get the directory listing from the remote host.

FTP (continued) (page-3)

Commands: (continued)

mdelete	- delete multiple files.
mdir	- 'list' the contents of multiple remote directories.
mget	- get multiple files using the default file transfer type.
mkdir	- make a directory on the remote machine.
mls	- 'nlist' the contents of multiple remote directories.
mode	- set the file transfer mode, the only valid mode currently is 'stream'.
mput	- send multiple files using the default file transfer type.
noninteractive	- turn off the prompting for multiple commands.
open	- open a connection to a remote host, only one connection is allowed at a time. If auto-login is on then will prompt for the user-name and password.
prompt	- toggles the interactive prompting for the multiple commands.
put	- send a file using the default file transfer type.
pwd	- print the current working directory for the remote host.
quit	- terminate the 'ftp' session and exit the 'ftp' program.
quote	- send an arbitrary ftp command.
recv	- get a file using the default file transfer type.
remotehelp	- get help (list of commands) from the remote server.

FTP (continued) (page-4)

Commands: (continued)

rename	- rename a file on the remote host.
rm	- remove a file on the remote host.
rmdir	- remove a directory on the remote host.
send	- put a file using the default file transfer type.
sendport	- toggles the use of the 'PORT' command for each data connection.
slashflip	- toggles the changing of '\' to '/' for outgoing commands.
status	- show the current status of such things as file transfer mode, file transfer type, and other general status.
struct	- set the file transfer structure, the only valid structure is 'file'.
type	- set the file transfer type, only 'ascii' and 'binary' are valid.
user	- send new user information, will request a password if needed.
verbose	- toggle the verbose mode on and off.

HOSTID

hostid [new_id]

This program is used to display and set the kernel 'hostid' value. Only the 'super-user' can change the 'hostid'. If no argument is specified, the current value is displayed, otherwise the argument is taken as the new value. For most systems the 'hostid' value has no function.

HOSTNAME

hostname [new_name]

This program is used to display and set the kernel 'hostname' value. Only the 'super-user' can change the 'hostname'. If no argument is specified, the current value is displayed, otherwise the argument is taken as the new value.

NETSTAT

netstat [-Aaihmnrst] [-f family] [-p proto] [-I interface]
[interval] [system] [core]

The 'netstat' program is used to display various information about the networking system. The default display is to show the current open connections.

Flags:

A - for displays which list connections, show the address of the kernel protocol control blocks for each connection.

a - for displays which list connections, show the servers which are awaiting connections, normally not shown.

i - instead of showing the connections, show the state of the various interfaces, including interface statistics.

h - instead of showing the connections, show the state of the IMP host tables (NOT IMPLEMENTED)

m - instead of showing the connections, show statistics about the network memory management system.

n - for all displays, instead of trying to convert network address to names, using the 'hosts' and 'networks' files, show them as numbers.

r - instead of showing the connections, show the current kernel routing tables.

s - instead of showing the connections, show statistics of the various protocol layers.

t - for the interface display (-i), show the status of each inteface watchdog timer. (NOT NORMALLY USED)

f - used to limit displays to a specific address family, the desired family must be given (AF_INET, AF_UNIX).

p - used to limit displays to a specific protocol, the desired protocol must be given (tcp, udp, ip, icmp).

I - used to limit displays to a specific interface, the desired interface must be given (et0, sl0, ...).

Only one of the flags in the set 'ihmrs' should be used at a time.

NETSTAT (continued) (page-2)

If both the 'r' and 's' flags are given, then the routing statistics are shown.

If the 'interval' parameter is specified, the program does not exit after showing the desired display once. It instead shows the display, with new values from the kernal, every 'interval' seconds. The 'interval' parameter only works with the 'interface' display.

The 'system' parameter allows you to change the system file that is used to determine the kernel addresses. The default is to use '/idris'. The selected file must have an intact symbol table, in order for 'netstat' to get the kernel addresses.

The 'core' parameter allows you to change the file that the data for the various displays is read from. The default is to use '/dev/kmem'. If the 'system' and 'core' files are not compatible, the results are unpredictable.

USER PROGRAMS

PING

ping [-dfnqrvR] host [packetsize [count [preload]]]

The 'ping' program is used to test and monitor network links and hosts.

Flags:

- d - turn on socket level debugging.
- f - enable the ping 'flood' option.
- n - set for numeric addresses only.
- q - enable quiet mode, don't show each reply.
- r - turn off socket level routing, send direct to address.
- v - enable verbose output.
- R - turn on the socket level 'Record Route' option.

The 'host' parameter is used to specify the host to send the 'echo' requests to.

The optional 'packetsize' paramter is used to specify the data size for the 'echo' request packets. If it is not specified, the default data size is 56 bytes.

The optional 'count' parameter is used to specify the number of 'echo' request packets to send. If the 'count' parameter is given, the 'packetsize' parameter must also be given. If the 'count' parameter is not given, ping sends 'echo' request continuously until the program is interrupted.

The optional 'preload' parameter is used to specify the number of 'echo' request packets to send before any replies are looked for. If the 'preload' parameter is given, the 'packetsize' and 'count' paramters must also be given.

When the 'ping' program is exited, statistics about the path to the specified host are displayed.

TELNET

telnet [-d] [-n filename|-] [host]

This program is used to obtain a teminal session on a remote host.

Flags:

d - enable socket level debug.

n - allows a file to specified where trace output goes to, such as that from 'show option' messages. The default is to use standard out.

The optional 'host' parameter is used to specify an initial host to try and connect to. If no host is specified then the program enters the command mode. If the host is specified, but the connection can't be made, after trying for some time, then a failure message is printed and the program enters the command mode.

Commands:

?	- print help information, either general, or for a specified command.
!	- escape to a local interactive shell, or if a command is given, pass it to a local shell, then return to command mode when it is done.
close	- close current connection, stay in telnet command mode.
display	- display current operating parameters.
help	- print help information, either general, or for a specified command.
mode	- set line-by-line or character-at-at-time mode, the valid arguments are 'line' and 'character'.
open	- try to connect to the specified host.
quit	- if connected, close the connection, in any case exit 'telnet'.

TELNET (continued) (page-2)

Commands: (continued)

send - transmit special characters, valid arguments are:

```
ao     - Send Telnet  Abort Output.
ayt    - Send Telnet  'Are You There'.
brk    - Send Telnet  Break.
ec     - Send Telnet  Erase Character.
el     - Send Telnet  Erase Line.
escape - Send current escape character.
ga     - Send Telnet 'Go Ahead' sequence.
ip     - Send Telnet Interrupt Process.
nop    - Send Telnet 'No operation'.
synch  0 Perform Telnet 'Synch operation'.
?      - Display send options.
```

set - set operating parameters, valid arguments are:

```
echo   - set character to toggle local echoing
         on/off.
escape - set character to escape back to the
         telnet command mode.
```

These need 'localchars' to be toggled to true:

```
erase  - set character to cause an Erase
         Character.
kill   - set character to cause an Erase Line.
?      - display help information.
```

status - print current 'telnet' status information.

TELNET (continued) (page-3)

Commands: (continued)

toggle - toggle operating parameters, valid arguments are:

autoflush	- toggle flushing of output when sending interrupt characters.
autosynch	- toggle automatic sending of interrupt characters when in urgent mode.
binary	- toggle sending and receiving of binary data.
crlf	- toggle sending carriage returns as telnet <CR><LF>.
crmod	- toggle mapping of the received carriage returns.
localchars	- toggle local recognition of certain control characters.
debug	- (debugging) toggle debugging.
netdata	- (debugging) toggle printing of hexadecimal network data.
options	- (debugging) toggle viewing of options processing.
?	- display help information.

DSTADDR

dstaddr interface dest_addr

This program is used to set the remote address of a 'slip' connection. The 'interface' argument is the name of the device such as 'sl0'. The 'dest_addr' argument may be specified as either a numeric address or a valid hostname from the 'hosts' file.

IFCONFIG

```
ifconfig interface
          [af [address [dest_addr]] [up] [down] [netmask mask]]
          [metric n] [trailers|-trailers] [arp|-arp]
          [debug|-debug] [broadcast addr] [ipdst addr]
```

This program is used to start an interface.

Caution: On some machines, trying to start an interface which is not installed will 'panic' the kernal. This is because the access to the non-existant device will generate a buss-error. Because of this danger, this program should be restricted to the 'super-user' only.

Normally the 'ifconfig' program is only run from '/adm/init'.

If just the 'interface' argument is specified, the current status of the specified 'interface' will be displayed.

Arguments:

up	- enable the interface, mark it as 'up'.
down	- disable the interface, mark it as 'down'.
trailers	- use trailer encapsulation on the interface, will only work with other hosts that support this. Trailer use on is the default mode.
-trailers	- disable trailer encapsulation on the interface, this must often be used with other network implementations.
arp	- enable the use of the Address Resolution Protocol, this is the default mode.
-arp	- disable the use of the Address Resolution Protocol.
debug	- enable kernel level debugging code for the specified interface. This may or may not be used by the interface.
-debug	- disable kernel level debugging code for the specified interface. This may or may not be used by the interface. This is the default.
netmask	- used to specify the mask that will be used to determine the network part of an address.

IFCONFIG (continued) (page-2)

Arguments: (continued)

metric - used to specify the routing metric for the interface. The higher the number, the more expensive the route through the interface.

broadcast - used to specify the broadcast address for the interface. The default broadcast address is for all bit positons to be set to '1'.

ipdst - only for XNS, not implemented.

The 'af' parameter allows an 'address family' to be specified. Since only the 'inet' family is currently allowed, it may be omitted. The address of the interface is usually the local hostname. The destination address of the interface is not needed for 'ethernet' interfaces, and is normally set with the 'dstaddr' command instead for 'slip' interfaces.

INETD

inetd init_file log_file bin_path

This program is started in '/adm/init'. It sets up to accept connections for the services specified in the 'init_file'. As connections are accepted for these services, the appropriate daemons are run to provide the services. The ports for the services are found in the file '/adm/services'.

Connection events are logged in the specified 'log_file.

The 'daemons' are looked for in the directory specified by 'bin_path'.

The standard 'init_file' is '/adm/inetd.conf'.

The standard 'log_file' is '/adm/inetd.log'.

The standard 'bin_path' is '/usr/local/lib'.

ROUTE

route [-n] [-f] [add|delete|change [net|hosts] args]

Flags:

n - display host and network names numerically, don't use names.

f - flush the current routing tables, if other commands are specified, then do this first.

Arguments:

add - add a new route, format is:

route add [net|host] destination gateway metric

the net/host parameter forces the interpetation of the destination parameter.

delete - delete a route, format is:

route delete [net|host] destination gateway

change - change a route (CURRENTLY NOT SUPPORTTED).

SLATTACH

slattach ttyname [baudrate]
    The default baudrate is 9600.

This program is used to attach a rs232 port to a 'slip' device. 'SLIP' stands for 'Serial-Line-IP', it provides a point-to-point connection between two hosts over an rs232 connection.. The system is normally configured for 4 'slip' devices. They are allocated in order from 'sl0' to 'sl3', with the first free one being attached. The 'slattach' program should be run in the background since it never exits. This is so the specified tty device is kept open. If the 'slattach' program is killed, the device will be closed, and it will become detached from the 'slip' device. While a tty device has a 'slattach' running on it, it can no-longer be read or written. After attaching the device, the 'dstaddr' and 'ifconfig' programs should be run to configure both ends of the 'slip' connection.

SYSTEM DAEMONS

FTPD

ftpd [-vdlx] [-t#]

This is the daemon loaded by 'inetd' to support remote 'ftp' sessions. For each remote 'ftp' session a separate copy of 'ftpd' is run.

Flags:

- v - turn debug on
- d - another way to turn debug on
- l - turn logging on
- x - turn 'yacc' debugging on
- t - (timeout time in seconds), this paramter specifies the amount of time that can pass with no activity before the connection is closed. The default timeout is 900 seconds.

SYSTEM DAEMONS

TELNETD

telnetd

This is the daemon loaded by 'inetd' to support remote terminal sessions. For each remote terminal session a separate copy of 'telnetd' is run.

SPECIAL FILES

/adm/inetd.conf (standard name, is really argument to 'inetd')

format of each line:

<service> <protocol> <daemon name> [arg1] [arg2] [arg3] [arg4]

Comment lines and blank lines are allowed, comment lines should begin with the '#' character.

/adm/inetd.log (standard name, is really argument to 'inetd')

This file is created by 'inetd', it adds an entry each time a new connection is accepted from a remote client.

SPECIAL FILES

/adm/hosts

format of each line:

<internet address> <hostname> [# comment]

Comment lines and blank lines are allowed, comment lines should begin with the '#' character. Comments may also appear at the end of each line, again preceded by the '#' character.

/adm/networks

format of each line:

<network-name> <network-number>

Comment lines and blank lines are allowed, comment lines should begin with the '#' character. Comments may also appear at the end of each line, again preceded by the '#' character.

/adm/protocols

format of each line:

<protocol-name> <protocol-number>

Comment lines and blank lines are allowed, comment lines should begin with the '#' character. Comments may also appear at the end of each line, again preceded by the '#' character.

/adm/services

format of each line:

<service-name> <port-number>/<protocol-name>

Comment lines and blank lines are allowed, comment lines should begin with the '#' character. Comments may also appear at the end of each line, again preceded by the '#' character.

SPECIAL FILES

/adm/ftpusers

This file contains a list of all user names to be disallowed 'ftp' access, such as 'uucp'.

format of each line:

<user-name>

There must be nothing else on the line if a match is to be found, the name must start in column 0 and be immediately followed by a linefeed. Comments may be placed on separate lines, and though no required, should be preceded by a '#' character for future compatibility.

SPECIAL FILES

/adm/init (must have network startup added)

/adm/.login (must setup user enviorment for telnet logins)
(path to network programs)

/adm/.logout

/(user)/.login (must setup user enviorment for telnet logins)
(path to network programs)

/(user)/.logout

/adm/passwd (must have entries for telnet and ftp users)

/idris (must be the booted kernel or linked to it, must still have its symbol table for 'netstat'.)

SPECIAL DEVICES

/dev/socket/sock(xx)

These devices are only used internally by the network interface library and should never be accessed directly.

/dev/select/sel(xx)

These devices are only used internally by the network interface library and should never be accessed directly.

/dev/pty[mnop][0123456789abcdef]

These are the master side of the 'puesdo-terminal' devices. They are mainly used by the 'telnetd' program to provide remote login capability.

The device driver only allows one user to have each of these devices open at a time. If another user tries to open a master 'pty' which is already open, the open will fail. This feature can be used to locate a free 'puesdo-device' pair. If a 'pty' is needed, an open is attempted on each master device in numerically increasing sequence till the open succeeds. The standard for device naming is:

ptym0-ptymf, ptyn0-ptynf, ... ptyp0-ptypf

Each master side of a 'puesdo-terminal' devices is connected via a bi-directional channel to a slave device, whose name can be easily determined by replacing the 'pty' part of the master name with 'tty'. There is a one-for-one correspondence between each master and slave device.

Normally only 'ptym0-ptymf' are implemented, for a total of 16 devices, or 16 remote logins maximum. This can only be changed by changing equates in the 'pty' device driver (really 'param.h') and recompiling. Then the additional devices must be created in the '/dev' directory.

/dev/tty[mnop][0123456789abcdef]

These are the slave side of the 'puesdo-terminal' devices. They are mainly used by the 'telnetd' program to provide remote login capability.

These devices attempt to look as much as possible like real external character devices (ie. RS232 ports). This is so programs such as 'sh' and 'log' will work correctly.

ACCEPT

```
int
accept(fd, name, anamelen)
    int              fd       ;
    struct sockaddr *name     ;
    int             *anamelen ;
```

The last step in accepting a connection is the 'accept' call. First the socket is created using the 'socket' call, then the socket is assigned an address with the 'bind' call, then it allows connection requests with the 'listen' call, then finally the connection is completed with the 'accept' call.

If no connections are pending on the socket, then 'accept' will block, waiting for a connection request' unless it has be set non-blocking.

After 'listen' has been called the 'select' function can be used to test the file-descriptor for 'reading' to see if there are any pending connections.

The 'address' parameter must point to space large enough to hold the address for the type of socket being referenced. Both the 'address' and 'anamelen' locations are filled with the address and len of the address of the connecition requestor.

If the 'accept' call succeeds, then it returns a file-descriptor which is greater-than or equal to zero. If it fails then negative-one is returned and 'errno' is set with an error-code.

The returned file-descriptor points to a new socket of the same type as the original socket. It is the new socket that is connected to the connection requestor's socket. The original socket is left open so that more connections can be accepted.

BCMP/BCOPY/BZERO

```
int
bcmp(s1, s2, length)
    char *s1     ;
    char *s2     ;
    int   length ;
```

This function compares the buffers at 's1' and 's2' for 'length' bytes. It returns zero if the the two buffers are the same or non-zero if they are different.

```
void
bcopy(src, dst, length) ;
    char *src    ;
    char *dst    ;
    int   length ;
```

This function copies the buffer at 'src' to 'dst' for 'length' bytes.

```
void
bzero(dst, length) ;
    char *dst    ;
    int   length ;
```

This function fills the buffer starting at 'dst' for 'length' bytes, with the constant value of zero.

BIND

```
int
bind(fd, name, namelen)
    int              fd      ;
    struct sockaddr *name    ;
    int              namelen ;
```

The 'bind' call is used to assign a 'name' (address) to a socket. The type of address and its length will depend on the address-family that the socket was created with.

In some implementations when 'bind' is called for a 'unix' domain socket the address actually appears in the file-system, that is not true of this implementation. Internal 'inodes' are used for 'unix' domain linkup instead of real file-system 'inodes.

If the 'bind' call succeeds, then it returns zero. If it fails then negative-one is returned and 'errno' is set with an error-code.

BYTEORDER

```
unsigned short
htons(x)
    unsigned short x ;

unsigned long
htonl(x)
    unsigned long x ;

unsigned short
ntohs(x)
    unsigned short x ;

unsigned long
ntohl(x)
    unsinged long x ;
```

These functions convert between 'network' and 'host' byte order for short (16-bit) and long (32-bit) integers.

htons - convert 16-bit 'host' value to 'network' order.

htonl - convert 32-bit 'host' value to 'network' order.

ntohs - convert 16-bit 'network' value to 'host' order.

ntohl - convert 32-bit 'network' value to 'host' order.

CONNECT

```
int
connect(fd, name, namelen)
    int              fd      ;
    struct sockaddr *name    ;
    int              namelen ;
```

The 'connect' function is used to request a connection to another socket which is 'listening' for a connection.

For 'SOCK_DGRAM' sockets this binds a permanent address to the socket.

The size of the address parameter 'name' is dependent on the address-family being used.

A client programm wishing to connect to a server will generally do a 'socket' call followed by a 'connect' call. The original file-descriptor returned by the socket call is connected to the other end of the connection, unlike 'accept' which returns a new file-descriptor.

If the 'connect' call succeeds, then it returns zero. If it fails then negative-one is returned and 'errno' is set with an error-code.

Unless the file-descriptor is marked non-blocking, the 'connect' call blocks for a 'long' time (minutes) trying to make the connection. If a connection can't be made it eventually times out and returns an error. If the socket is marked non-blocking the 'select' call can check for 'writing' to tell when the connection has been completed.

GETCWD

```
char *
getcwd(dst, length)
    char *dst    ;
    int   length ;
```

This function returns the current working directory. The path is returned in the specified buffer at 'dst', up to the maximum 'length' specified. The returned pathname will be null-terminated, so the max returned pathname lenght is 'length' - 1.

This function is currently in the library to get around bugs in the present standard 'C' library calls. It is implemented by calling the 'pwd' program, with its output directed to a pipe.

GETDTABLESIZE

```
int
getdtablesize()
```

This function returns the number of available file descriptors on the system, for each user. This is the maximum number of open files that each user can have.

GETHOSTID/SETHOSTID

```
long
gethostid()

int
sethostid(id)
    long id ;
```

These functions get and set the local host hostid. A negative value is returned if these functions fail. On sucess 'gethostid' returns the current 'hostid' value. The 'sethostid' value returns zero if the function completes ok. Only the 'super-user' may change the 'hostid'. The 'hostid' value is generally not used by any programs and serves no purpose on most systems.

GETHOSTNAME/SETHOSTNAME

```
int
gethostname(name, namelen)
    char *name    ;
    int   namelen ;

int
sethostname(name, namelen)
    char *name    ;
    int   namelen ;
```

These functions get and set the local 'hostname'. The address and length of the buffer are passed to the functions. A zero value is returned if the function suceeds, otherwize non-zero is returned and 'errno' contains the reason for the failure. Only the 'super-user' may change the 'hostname'. Most network programs use the 'gethostname' function to determine the local 'hostname'.

GETPEERNAME

```
int
getpeername(fd, asa, alen)
    int              fd   ;
    struct sockaddr *asa  ;
    int             *alen ;
```

This function is used to get the address of the remote end of a connected socket, specified by the passed 'fd'. On success, the structure referenced by 'asa' is filled out with the remote address of the socket, and the length of the address is returned at 'alen'.

If the 'getpeername' call succeeds, then it returns zero. If it fails then negative-one is returned and 'errno' is set with an error-code.

GETSOCKNAME

```
int
getsockname(fd, asa, alen)
    int              fd   ;
    struct sockaddr *asa  ;
    int             *alen ;
```

This function is used to get the address bound to a local socket specified by the passed 'fd'. On success, the structure referenced by 'asa' is filled out with the address of the socket, and the length of the address is returned at 'alen'.

If the 'getsockname' call succeeds, then it returns zero. If it fails, then negative-one is returned and 'errno' is set with an error-code.

GETSOCKOPT/SETSOCKOPT

```
int
getsockopt(fd, level, name, val, avalsize)
    int      fd       ;
    int      level    ;
    int      name     ;
    caddr_t *val      ;
    int     *avalsize ;

int
setsockopt(fd, level, name, val, valsize)
    int      fd      ;
    int      level   ;
    int      name    ;
    caddr_t *val     ;
    int      valsize ;
```

These functions are used to set and get the current status of settable socket options.

If the 'getsockopt' or 'setsockopt' calls succeed, then they returns zero. If the call fails, then negative-one is returned and 'errno' is set with an error-code.

GETTIMEOFDAY

```
int
gettimeofday(tp, tzp)
    struct timeval  *tp  ;
    struct timesone *tzp ;
```

Currently 'tzp' returns no usefull value, function only implemented for the timing needed in the 'ping' command where one second resolution is not good enough. The 'tp' pointer should point to a 'struct timeval' which will be filled in on successful return. The function returns a zero if successful and a negative one if it fails. It will fail if 'tp' is NULL. It is ok for 'tzp' to be NULL.

HOSTENT

```
void
endhostent()

void
sethostent(stayopen)
    int stayopen ;

struct hostent *
gethostent()

struct hostent *
gethostbyname(name)
    char *name ;

struct hostent *
gethostbyaddr(addr, len, type)
    char *addr ;
    int   len  ;
    int   type ;

struct hostent
    {
    char  *h_name      ; /* official name of the host          */
    char **h_aliases   ; /* alias list (for now always empty)  */
    int    h_addrtype  ; /* host address type (always AF_INET) */
    int    h_length    ; /* length of an address (sizeof(long))*/
    char **h_addr_list ; /* list of address (for now only one) */
    } ;
#define h_addr h_addr_list[0] ;  /* for backward compatibility */
```

These functions are used to retreive information from the 'host' database (/adm/hosts). The structure 'hostent' is defined in 'netdb.h'. The functions that return a pointer to a 'hostent' structure return a pointer to a 'static' data location.

endhostent - if the 'host' database is open, close it.

sethostent - sets a flag that if non-zero leaves the database open accross multiple calls to 'gethostbyname' and 'gethostbyaddr'. If the flag is zero then the 'host' database is closed after each of these two calls.

gethostent - gets the next 'host' entry entry from the database, if the database is closed, it opens it and returns the first entry.

gethostbyname - returns a pointer to the host entry that matches the specified host name. If no matching entry is found then a 'NULL' pointer is returned. If the database is closed, it is opened. In any case the search always starts at the first entry. If 'stayopen' was set then the database is left open at the end of the search, otherwise it is closed.

gethostbyaddr - returns a pointer to the host entry that matches the specified host address, length, and type.. If no matching entry is found then a 'NULL' pointer is returned. If the database is closed, it is opened. In any case the search always starts at the first entry. If 'stayopen' was set then the database is left open at the end of the search, otherwise it is closed.

INDEX/RINDEX

```
char *
index(src, c)
    char *src ;
    char  c   ;
```

This function is the same as 'strchr(src, c)'. It returns the address of the first occurance of the character 'c' in the string at 'src'. If the character 'c' is not in the 'src' string the address of the terminating '\0' is returned.

```
char *
rindex(src, c)
    char *src ;
    char  c   ;
```

This function is the same as 'strrchr(src, c)'. It returns address of the last occurance of the character 'c' in the string at 'src'. If the character 'c' is not in the 'src' string the address of the terminating '\0' is returned.

In general 'macros' may be used for these functions, but sometimes it is just easier to not have to change the source-code and instead link to these functions.

The following 'macros' can be used:

```
#define  index(a,b) strchr(a,b)
#define rindex(a,b) strrchr(a,b)
```

INET

```
unsigned long
inet_addr(name)
    char *name ;
```

This function parses a decimal string in the form of 'xxx.xxx.xxx.xxx' and returns the internet address in network byte ordering, not necessarily machine byte order.

```
unsigned long
inet_network(name)
    char *name ;
```

```
char *
inet_ntoa(in)
    struct in_addr in ;
```

This functions returns the ascii representation of an Internet address.

```
int
inet_netof(in)
    struct in_addr in ;
```

This function returns the 'network number' part of an Internet address.

```
int
inet_lnaof(in)
    struct in_addr in ;
```

This function returns the 'local network address' part of an Internet address.

```
struct in_addr
inet_makeaddr(net, ina)
    unsigned long net, ina ;
```

This function takes the 'network number' and 'local network address' and returns a Internet address.

LISTEN

```
int
listen(fd, backlog)
    int fd      ;
    int backlog ;
```

If it is desired to 'accept' connections on a socket of type 'SOCK_STREAM' or 'SOCK_DGRAM' the 'listen' call must be made on the socket. The 'backlog' parameter defines how pending connections can be waiting on a given socket. This value is currently limited to five.

If the 'listen' call succeeds, then it returns zero. If it fails then negative-one is returned and 'errno' is set with an error-code.

NETENT

```
void
endnetent()

void
setnetent(stayopen)
    int stayopen ;

struct netent *
getnetent()

struct netent *
getnetbyname(name)
    char *name ;

struct netent *
getnetbyaddr(net, type)
    long net  ;
    int  type ;

struct netent
    {
    char         *n_name     ; /* officail network name       */
    char        **n_aliases  ; /* alias list (always empty)   */
    int           n_addrtype ; /* net address type            */
    unsigned long n_net      ; /* network (must be < 32-bits) */
    } ;
```

These functions are used to retreive information from the 'network' database (/adm/networks). The structure 'netent' is defined in 'netdb.h'. The functions that return a pointer to a 'netent' structure return a pointer to a 'static' data location.

endnetent - if the 'network' database is open, close it.

setnetent - sets a flag that if non-zero leaves the database open accross multiple calls to 'getnetbyname' and 'getnetbyaddr'. If the flag is zero then the 'network' database is closed after each of these two calls.

getnetent - gets the next 'network' entry from the database, if the database is closed, it opens it and returns the first entry.

NETENT (continued)

getnetbyname - returns a pointer to the 'network' entry that matches the specified 'network' name. If no matching entry is found then a 'NULL' pointer is returned. If the database is closed, it is opened. In any case the search always starts at the first entry. If 'stayopen' was set then the database is left open at the end of the search, otherwise it is closed.

getnetbyaddr - returns a pointer to the 'network' entry that matches the specified 'network' number and type. If no matching entry is found then a 'NULL' pointer is returned. If the database is closed, it is opened. In any case the search always starts at the first entry. If 'stayopen' was set then the database is left open at the end of the search, otherwise it is closed.

NLIST

```
void
nlist(system, nl)
    char         *system ;
    struct nlist *nl     ;
```

This function is used to get the addresses of specified labels from an object file. Often is is used to get addresses from '/idris' which are then used to access the actual location in the running kernel through '/dev/kmem'. The name of the object file is passed as 'system', which in the stated example would be '/idris'. The 'struct nlist' would be filled with the names of the label for which the addresses are wanted. The address elements of the structure are left blank before calling the 'nlist' function. Upon return from the function the address fields will have been filled in or set to NULL if the label was not found.

PERROR

```
char *
perror(message)
    char *message ;

int sys_nerr ;
char *sys_errlist[] ;
extern int errno ;
```

This functions finds the text strings associated with the current value of 'errno'. If 'message' is equal to 'NULL' then the function returns a pointer to the string, otherwise it outputs 'message', followed by a colon, then a space, then the string, and finally a line-feed.

This function replaces the standard 'Idris' one since the 'network' code requires many more 'errno' values.

The 'sys_nerr' integer can be referenced to find how many entrys are in the error message table 'sys_errlist'. This is so the message table can be used other than through the 'perror' function.

PROTOENT

```
void
endprotoent()

void
setprotoent()
    int stayopen ;

struct protoent *
getprotoent()

struct protoent *
getprotobyname(name)
    char *name ;

struct protoent *
getprotobynumber(proto)
    int proto ;

struct protoent
    {
    char  *p_name    ; /* offical protocol name     */
    char **p_aliases ; /* alias list (always empty) */
    int    p_proto   ; /* protocol number           */
    } ;
```

These functions are used to retreive information from the 'protocol' database (/adm/protocols). The structure 'protoent' is defined in 'netdb.h'. The functions that return a pointer to a 'protoent' structure return a pointer to a 'static' data location.

endprotoent - if the 'protocol' database is open, close it.

setprotoent - sets a flag that if non-zero leaves the database open accross multiple calls to 'getprotobyname' and 'getprotobynumber'. If the flag is zero then the 'protocol' database is closed after each of these two calls.

getprotoent - gets the next 'protocol' entry from the database, if the database is closed, it opens it and returns the first entry.

PROTOENT (continued) (page-2)

getprotobyname - returns a pointer to the 'protocol' entry that matches the specified 'protocol' name. If no matching entry is found then a 'NULL' pointer is returned. If the database is closed, it is opened. In any case the search always starts at the first entry. If 'stayopen' was set then the database is left open at the end of the search, otherwise it is closed.

getprotobynumber - returns a pointer to the 'protocol' entry that matches the specified 'protocol' number. If no matching entry is found then a 'NULL' pointer is returned. If the database is closed, it is opened. In any case the search always starts at the first entry. If 'stayopen' was set then the database is left open at the end of the search, otherwise it is closed.

RECV/RECVFROM/RECVMSG

```
int
recv(fd, buf, len, flags)
    int   fd    ;
    char *buf   ;
    int   len   ;
    int   flags ;

int
recvfrom(fd, buf, len, flags, from, fromlen)
    int              fd      ;
    char            *buf     ;
    int              len     ;
    struct sockaddr *from    ;
    int             *fromlen ;

int
recvmsg(fd, msg, flags)
    int            fd    ;
    struct msghdr *msg   ;
    int            flags ;

struct msghdr                       /* defined in 'sys/socket.h' */
{
    caddr_t       msg_name     ; /* optional address or NULL  */
    int           msg_nameline ; /* address length if address */
    struct iovec *msg_iov      ; /* data vector               */
    int           msg_iovlen   ; /* data vector length        */
    caddr_t       msg_accrights ; /* access rights sent/recvd */
    int           msg_accrightslen ; /* access rights length  */
} ;
```

These functions are used to receive data over a socket. The standard 'read' call may also be used. All of these calls normally block if no more data is available, unless the socket is marked non-blocking. The 'select' call can be used to tell if the more data is available on the socket.

The 'recvfrom' and 'recvmsg' calls must be used if no address has been bound to a 'SOCK_DGRAM' socket. The 'recv' and 'read' calls require a 'connected' socket. It the address pointers are not NULL then the referenced space is filled with the senders address.

If the 'recv' calls succeed, then they return the number of bytes received. If they fail then negative-one is returned and 'errno' is set with an error-code.

SELECT

```
int
select(max_fd, rd_mask, wr_mask, ex_mask, ptime)
    int             max_fd  ;
    unsigned long  *rd_mask ;
    unsigned long  *wr_mask ;
    unsigned long  *ex_mask ;
    struct timeval *ptime   ;
```

The 'select' call is used to check a number of file-descriptors at one for their ability to be read or written to. Also exception conditions may be checked for.

Each of the three mask pointers (rd_mask, wr_mask, ex_mask), points to a vector of longs which have bits set that represent file-descriptors. Since currently only 32 file-descriptors are allowed per process, each vector contains only a single entry. If a vector pointer is set to NULL, that condition is not checked. To check if data is available on file-descriptor 5 you would set the 5th bit (from zero) of the long pointer to by 'rd_mask'. This can be done by:

```
mask |= 1 << 5 ; /* pass the address of mask to select */
```

The 'max_fd' entry should 1 greater that the highest file-descriptor that is to be checked. If 5 is the highest file-descriptor then 'max_fd' should be 6. This means only look at the first 6 bits of each mask.

The 'rd_mask' is used to check for data available, the 'wr_mask' checks if you can write without blocking, and the 'ex_mask' checks for exception conditions such as connections being broken.

If any of the conditions are satisfied then 'select' returns the number of conditions that were found satisfied and only the satisfied bits are left set in the various mask vectors.

If no conditions are satisfied the 'ptime' parameter controls what will happen. If 'ptime' is NULL then the call will block until at least one condition is satisfied. If 'ptime' points to a 'timeval' structure with the elements set to zero then the call returns immediately, and if no conditions are satisfied, a zero is returned. If the 'timeval' structure is non-zero, then the 'select' call will wait until that amout of time has passed before returning. It will immediately return once some conditions are satisfied or after the time has passed, whichever occurs first. Again it will either return the number of conditions met, or zero if none.

SELECT (continued)

If any errors occur, such as broken connections, then select will immediately return a negative one. To determine which file-descriptor caused the error condition it will be necessary to try each one separately. This can be done using a zeroed 'timeval' to keep the 'select' call from blocking from blocking.

SEND/SENDTO/SENDMSG

```
int
send(fd, buf, len, flags)
    int   fd    ;
    char *buf   ;
    int   len   ;
    int   flags ;

int
sendto(fd, buf, len, flags, to, tolen)
    int              fd    ;
    char            *buf   ;
    int              len   ;
    int              flags ;
    struct sockaddr *to    ;
    int              tolen ;

int
sendmsg(fd, msg, flags)
    int            fd    ;
    struct msghdr *msg   ;
    int            flags ;

struct msghdr                       /* defined in 'sys/socket.h' */
{
    caddr_t       msg_name     ; /* optional address or NULL  */
    int           msg_nameline ; /* address length if address */
    struct iovec *msg_iov      ; /* data vector               */
    int           msg_iovlen   ; /* data vector length        */
    caddr_t       msg_accrights ; /* access rights sent/recvd */
    int           msg_accrightslen ; /* access rights length  */
} ;
```

These functions are used to send data over a socket. The standard 'write' call may also be used. All of these calls normally block if buffer space is not available, unless the socket is marked non-blocking. Sucess of the calls does not imply that the data has reached the destination, only that it was buffered for transmission. The 'select' call can be used to tell if the calls will accept more data.

The 'sendto' and 'sendmsg' calls must be used if no address has been bound to a 'SOCK_DGRAM' socket. The 'send' and 'write' calls require a 'connected' socket.

If the 'send' calls succeed, then they return the number of bytes sent. If they fail then negative-one is returned and 'errno' is set with an error-code.

SERVENT

```
void
endservent()

void
setservent(stayopen)
    int stayopen ;

struct servent *
getservent()

struct servent *
getservbyname(name)
    char *name ;

struct servent *
getservbyport(port, proto)
    int   port  ;
    char *proto ;

struct servent
    {
    char  *s_name    ; /* offical service name      */
    char **s_aliases ; /* alias list (always empty) */
    int    s_port    ; /* port for service          */
    char  *s_proto   ; /* name of protocol to use   */
    } ;
```

These functions are used to retreive information from the 'service' database (/adm/services). The structure 'servent' is defined in 'netdb.h'. The functions that return a pointer to a 'servent' structure return a pointer to a 'static' data location.

endservent - if the 'service' database is open, close it.

setservent - sets a flag that if non-zero leaves the database open accross multiple calls to 'getservbyname' and 'getservbyport'. If the flag is zero then the 'service' database is closed after each of these two calls.

getservent - gets the next 'service' entry from the database, if the database is closed, it opens it and returns the first entry.

SERVENT (continued)

getservbyname - returns a pointer to the 'service' entry that matches the specified 'service' name. If no matching entry is found then a 'NULL' pointer is returned. If the database is closed, it is opened. In any case the search always starts at the first entry. If 'stayopen' was set then the database is left open at the end of the search, otherwise it is closed.

getservbyport - returns a pointer to the 'service' entry that matches the specified 'service' port and protocl. If no matching entry is found then a 'NULL' pointer is returned. If the database is closed, it is opened. In any case the search always starts at the first entry. If 'stayopen' was set then the database is left open at the end of the search, otherwise it is closed.

SETLINEBUF

```
void
setlinebuf(stream)
    FILE *stream ;
```

Currently this function does nothing but is needed by the network code. It is normally used to set the buffering for a stream file into a line oriented mode. This is mainly used for debug files and does not affect network operation.

SHUTDOWN

```
int
shutdown(fd, how)
    int fd  ;
    int how ;
```

This call is used to close all or part of a socket connection.

```
    how = 0     - stops receives on the socket
    how = 1     - stops sending  on the socket
    how = 2     - stops both receives and sending
```

If the 'shutdown' call succeeds, then it returns zero. If it fails then negative-one is returned and 'errno' is set with an error-code.

SOCKET

```
int
socket(domain, type, protocol)
    int domain   ;
    int type     ;
    int protocol ;
```

This function returns a file-descriptor by which a 'socket' can be referenced. The socket is attached to the specified 'domain', 'type' and 'protocol'. A socket is a communication endpoint that can read and written. The file-descriptor is also used by special control functions such as 'setsockopt' to control various options about the socket. Usually the sockets provide a bidirectional connection to another socket, on the same or different machine.

The following 'domains' are currently supported:

    AF_UNIX
    AF_INET

The following 'types' are currently supported:

    SOCK_STREAM
    SOCK_DGRAM
    SOCK_RAW

The following 'protocols' are currently supported:

    PF_UNIX
    PF_INET

The 'protocol' should match the 'domain'.

If the 'socket' call succeeds, then it returns a file-descriptor which is greater-than or equal to zero. If it fails then negative-one is returned and 'errno' is set with an error-code.

SOCKETPAIR

```
int
socketpair(domain, type, protocol, afds)
    int domain   ;
    int type     ;
    int protocol ;
    int afds[2]  ;
```

This call creates a pair of connecting sockets of the specified 'domain', 'type', and 'protocol'. It returns the pair of file-descriptors in 'afds[2]'. The two ends of the connection are exactly the same. There is no need to associate an address with these sockets, they can be used just like a pipe except they are bi-directional.

If the 'socketpair' call succeeds, then it returns zero. If it fails then negative-one is returned and 'errno' is set with an error-code.

SYSLOG

```
void
openlog(pgm_name, flags)
    char *pgm_name ; /* shows up in all 'syslog' entries */
    int   flags    ; /* not currently used               */

void
syslog(type, mesg, x0, x1, x2, x3)
    int  type  ; /* not used */
    char *mesg ;
    int   x0   ; /* optional */
    int   x1   ; /* optional */
    int   x2   ; /* optional */
    int   x3   ; /* optional */
```

The 'mesg' string may have 'sprintf' type format arguments in it, in which case up to 4 arguments may be passed to this function.

This function writes a message to 'stderr' which contains the current time, followed by the progrom name from 'openlog', then the formatted message. If a linefeed is desired at the end of the output in must be provided in the message.