```
   /$$$$$$   /$$     /$$  /$$$$$$$  /$$$$$$  /$$$$$$$
  /$$__  $$ /$$$$   /$$$$ | $$__  $$ /$$__  $$| $$__  $$
 |__/  \ $$|_  $$  |_  $$ | $$  \ $$| $$  \__/| $$  \ $$
   /$$$$$$/  | $$    | $$ | $$$$$$$ |  $$$$$$ | $$  | $$
  /$$____/   | $$    | $$ | $$__  $$ \____  $$| $$  | $$
 | $$        | $$    | $$ | $$  \ $$ /$$  \ $$| $$  | $$
 | $$$$$$$$ /$$$$$$ /$$$$$$| $$$$$$$/|  $$$$$$/| $$$$$$$/
 |_____/|_____/|_____/|_____/  _____/ |_____/


      ****************************************************
      ************** The 211BSD man page project **************
      ****************************************************
      ***************** 4 – Special Files ********************
      ****************************************************
```

Inspired by:
==
  SimH   http://simh.trailing-edge.com/
  PiDP11   https://obsolescence.wixsite.com/obsolescence/pidp-11
  BSD 2.11   https://wfjm.github.io/home/211bsd/

Presented by the ShadowTron Blog
==
  https://www.youtube.com/c/shadowtronblog
  www.shadowtron.com
  shadowtronblog <at> gmail <dot> com

Other manuals in the series
==
     Manual 1 - Commands and Application Programs
     Manual 2 - System Calls
     Manual 3 - C Library Subroutines
     Manual 3F - Fortran Library
==> Manual 4 - Special Files
     Manual 5 - File Formats
     Manual 6 - Games
     Manual 7 - Miscellaneous
     Manual 8 - System Maintenance

                                                        Version 3.0

```
Page Command Description
==== ======= =====================================================================
   4 intro   introduction to special files and hardware support
   7 acc     ACC LH/DH IMP interface
   8 arp     Address Resolution Protocol
   9 bk      line discipline for machine-machine communication
  12 br      EATON BR1537/BR1711 1538[A,B,C,D] moving head disk
  15 cons    VAX-11 console interface
  16 css     DEC IMP-11A LH/DH IMP interface
  17 de      DEC DEUNA 10 Mb/s Ethernet interface
  19 dh      DH-11/DM-11 communications multiplexer
  21 dhu     DHU-11 communications multiplexer
  23 dhv     DHV-11 communications multiplexer
  25 dmc     DEC DMC-11/DMR-11 point-to-point communications device
  27 dr      DR11-W general purpose interface driver
  29 dz      DZ-11 communications multiplexer
  31 ec      3Com 10 Mb/s Ethernet interface
  33 en      Xerox 3 Mb/s Ethernet interface
  36 fd      stdin, stdout,  stderr file descriptor files
  37 hk      RK6-11/RK06 and RK07 moving head disk
  39 ht      TM-03/TE-16, TU-45, TU-77 MASSBUS magtape interface
  41 hy      Network Systems Hyperchannel interface
  43 icmp    Internet Control Message Protocol
  44 idp     Xerox Internet Datagram Protocol
  46 il      Interlan NI1010 10 Mb/s Ethernet interface
  48 imp     IMP raw socket interface
  50 imp     1822 network interface
  52 inet    Internet protocol family
  54 ip      Internet Protocol
  56 lp      line printer
  58 mem     main memory
  59 mtio    UNIX magtape interface
  62 intro   networking - introduction to networking facilities
  67 ns      Xerox Network Systems(tm) protocol family
  69 nsip    software network interface encapsulating ns packets in ip packets
  71 null    data sink
  72 pty     pseudo terminal driver
  75 qe      DEC DEQNA Q-bus 10 Mb/s Ethernet interface
  76 ra      MSCP disk controller interface
  79 ram     ram disk driver
  81 rk      RK-11/RK05 disk
  83 rl      RL-11/RL01,  RL02 moving-head disk
  85 rb      rx,rz,rb,rz - XMODEM,YMODEM,ZMODEM (Batch) file receive
  90 rx      DEC RX02 floppy disk\
  93 si      SI 9500/CDC 9766 moving head disk
  96 spp     Xerox Sequenced Packet Protocol
  99 sri     DR11-C IMP interface
 100 swap    swap device
 101 tb      line discipline for digitizing devices
 103 tcp     Internet Transmission Control Protocol
 106 tm      TM-11/TE-10 magtape interface
 108 tmscp   DEC TMSCP magtape interface
 110 ts      TS-11 magtape interface
 112 tty     general terminal interface
```

```
127 udp       Internet User Datagram Protocol
129 vv        Proteon proNET 10 Megabit ring
131 xp        generic SMD moving-head disk
```

NAME
     intro - introduction to special files and hardware support

DESCRIPTION
     This section describes the special files, related driver
     functions, and networking support available in the system.
     In this part of the manual, the SYNOPSIS section of each
     configurable device gives a sample specification for use in
     constructing a system description for the
     /sys/conf/configscript, the autoconfig(8), program and des-
     cibes the major and minor device numbers and their encoding.
     The DIAGNOSTICS section lists messages which may appear on
     the console and/or in the system error log /usr/adm/messages
     due to errors in device operation.

     This section contains both devices which may be configured
     into the system, ``4'' entries, and network related informa-
     tion, ``4N'', ``4P'', and ``4F'' entries; The networking
     support is introduced in intro(4N).

PDP DEVICE SUPPORT
     This section describes the hardware supported on the DEC
     PDP-11. Software support for these devices comes in two
     forms. A hardware device may be supported with a character
     or block device driver, or it may be used within the net-
     working subsystem and have a network interface driver.
     Block and character devices are accessed through files in
     the file system of a special type; c.f. mknod(8). Network
     interfaces are indirectly accessed through the interprocess
     communication facilities provided by the system; see
     socket(2).

     A hardware device is identified to the system at configura-
     tion time and the appropriate device or network interface
     driver is then compiled into the system. When the resultant
     system is booted, the autoconfiguration facilities in the
     system probe for the device on either the UNIBUS (or Q-bus)
     or MASSBUS and, if found, enable the software support for
     it. If a device does not respond at autoconfiguration time
     it is not accessible at any time afterwards. To enable a
     device which did not autoconfigure, the system will have to
     be rebooted.

     The autoconfiguration system is described in autoconfig(8).
     A list of the supported devices is given below.

SEE ALSO
     intro(4), networking(4), config(8).

LIST OF DEVICES
     The devices listed below are supported in this incarnation
     of the system. Pseudo-devices are not listed. Listed also

are devices which are in various stages of porting to
2.11BSD from 4.3BSD.

Devices are indicated by their functional interface. If
second vendor products provide functionally identical inter-
faces they should be usable with the supplied software.
(Beware, however, that we promise the software works ONLY
with the hardware indicated on the appropriate manual page.)
Occasionally, new devices of a similar type may be added
simply by creating appropriate table entries in the driver.

The following are known to work:
acc   ACC LH/DH IMP communications interface
de   DEC DEUNA 10Mb/s Ethernet controller
dh   DH-11 emulators, terminal multiplexor
dhu   DHU-11 terminal multiplexor
dz   DZ-11 terminal multiplexor
ec   3Com 10Mb/s Ethernet controller
hk   RK6-11/RK06 and RK07 moving head disk
ht   TM03 MASSBUS tape drive interface (with TE-16, TU-45, TU-77)
il   Interlan 1010, 1010A, 2010A 10Mb/s Ethernet controller
lp   LP-11 parallel line printer interface
qe   DEC DEQNA Q-bus 10 Mb/s Ethernet interface
ra   DEC UDA-50, RQDX, KLESI disk controllers
rk   DEC RK05 disk controller
rl   DEC RL-11 disk controller
rx   DEC RX02 floppy interface
si   SI 9500 disk controller
tm   TM-11/TE-10 tape drive interface
tmscp   TMSCP-compatible tape controllers (e.g., TU81, TK50)
ts   TS-11 tape drive interface
vv   Proteon proNET 10Mb/s and 80Mb/s ring network interface
xp   General purpose SMD disk controller

The following should work:
dr   DR-11W general purpose DMA UNIBUS interface

The following worked in the past but will probably require work:
css   DEC IMP-11A communications interface
dmc   DEC DMC-11/DMR-11 point-to-point communications device
en   Xerox 3Mb/s Ethernet controller (obsolete)
sri   DR-11C IMP interface

It should be possible to port these from 4.3BSD:
ex   Excelan 10Mb/s Ethernet controller
ix   Interlan NP-100 10Mb/s Ethernet controller
np   Interlan NP-100 10Mb/s Ethernet controller (intelligent mode)
pcl   DEC PCL-11 communications interface

No idea whether any of the following could be ported:

ad	Data translation A/D interface
ct	C/A/T or APS phototypesetter
ddn	ACC ACP625 DDN Standard Mode X.25 IMP interface
dmf	DEC DMF-32 terminal multiplexor and parallel printer interface
dmz	DEC DMZ-32 terminal multiplexor
dn	DEC DN-11 autodialer interface
hdh	ACC IF-11/HDH IMP interface
hp	MASSBUS disk interface (with RP06, RM03, RM05, etc.)
hy	DR-11B or GI-13 interface to an NSC Hyperchannel
kg	KL-11/DL-11W line clock
mt	TM78 MASSBUS tape drive interface
tu	VAX-11/730 TU58 console cassette interface
un	DR-11W interface to Ungermann-Bass
up	Emulex SC-21V, SC-31 UNIBUS disk controller
ut	UNIBUS TU-45 tape drive interface
uu	TU58 dual cassette drive interface (DL11)
va	Benson-Varian printer/plotter interface
vp	Versatec printer/plotter interface

NAME
     acc - ACC LH/DH IMP interface

SYNOPSIS
     /sys/conf/SYSTEM:
        NACC 0   # ACC LH/DH ARPAnet IMP interface
        PLI YES # LH/DH is connected to a PLI

DESCRIPTION
     The acc device provides a Local Host/Distant Host interface
     to an IMP. It is normally used when participating in the
     DARPA Internet. The controller itself is not accessible to
     users, but instead provides the hardware support to the IMP
     interface described in imp(4). When configuring, the
     imp(NIMP) pseudo-device must also be included.

DIAGNOSTICS
     acc%d: not alive. The initialization routine was entered
     even though the device did not autoconfigure. This indi-
     cates a system problem.

     acc%d: can't initialize. Insufficient UNIBUS resources
     existed to initialize the device. This is likely to occur
     when the device is run on a buffered data path on an 11/750
     and other network interfaces are also configured to use buf-
     fered data paths, or when it is configured to use buffered
     data paths on an 11/730 (which has none).

     acc%d: imp doesn't respond, icsr=%b. The driver attempted
     to initialize the device, but the IMP failed to respond
     after 500 tries. Check the cabling.

     acc%d: stray xmit interrupt, csr=%b. An interrupt occurred
     when no output had previously been started.

     acc%d: output error, ocsr=%b, icsr=%b. The device indicated
     a problem sending data on output.

     acc%d: input error, csr=%b. The device indicated a problem
     receiving data on input.

     acc%d: bad length=%d. An input operation resulted in a data
     transfer of less than 0 or more than 1008 bytes of data into
     memory (according to the word count register). This should
     never happen as the maximum size of a host-IMP message is
     1008 bytes.

NAME
     arp - Address Resolution Protocol

SYNOPSIS
     /sys/conf/SYSTEM:
        NETHER   1   # ether pseudo-device

DESCRIPTION
     ARP is a protocol used to dynamically map between DARPA
     Internet and 10Mb/s Ethernet addresses. It is used by all
     the 10Mb/s Ethernet interface drivers. It is not specific
     to Internet protocols or to 10Mb/s Ethernet, but this imple-
     mentation currently supports only that combination.

     ARP caches Internet-Ethernet address mappings. When an
     interface requests a mapping for an address not in the
     cache, ARP queues the message which requires the mapping and
     broadcasts a message on the associated network requesting
     the address mapping. If a response is provided, the new
     mapping is cached and any pending message is transmitted.
     ARP will queue at most one packet while waiting for a map-
     ping request to be responded to; only the most recently
     ``transmitted'' packet is kept.

     To facilitate communications with systems which do not use
     ARP, ioctls are provided to enter and delete entries in the
     Internet-to-Ethernet tables. Usage:

        #include <sys/ioctl.h>
        #include <sys/socket.h>
        #include <net/if.h>
        struct arpreq arpreq;

        ioctl(s, SIOCSARP, (caddr_t)&arpreq);
        ioctl(s, SIOCGARP, (caddr_t)&arpreq);
        ioctl(s, SIOCDARP, (caddr_t)&arpreq);
     Each ioctl takes the same structure as an argument.
     SIOCSARP sets an ARP entry, SIOCGARP gets an ARP entry, and
     SIOCDARP deletes an ARP entry. These ioctls may be applied
     to any socket descriptor s, but only by the super-user. The
     arpreq structure contains:

        /*
         * ARP ioctl request
         */
        struct arpreq {
              struct sockaddr arp_pa; /* protocol address */
              struct sockaddr arp_ha; /* hardware address */
              int   arp_flags;/* flags */
        };
        /* arp_flags field values */
        #define ATF_COM   0x02/* completed entry (arp_ha valid) */

Printed 6/17/012 August 1, 1987   1

```
      #define   ATF_PERM        0x04     /* permanent entry */
      #define   ATF_PUBL        0x08     /* publish (respond for other host) */
      #define   ATF_USETRAILERS  0x10     /* send trailer packets to host */
```

The address family for the arp_pa sockaddr must be AF_INET;
for the arp_ha sockaddr it must be AF_UNSPEC. The only flag
bits which may be written are ATF_PERM, ATF_PUBL and
ATF_USETRAILERS. ATF_PERM causes the entry to be permanent
if the ioctl call succeeds. The peculiar nature of the ARP
tables may cause the ioctl to fail if more than 8 (per-
manent) Internet host addresses hash to the same slot.
ATF_PUBL specifies that the ARP code should respond to ARP
requests for the indicated host coming from other machines.
This allows a host to act as an ``ARP server,'' which may be
useful in convincing an ARP-only machine to talk to a non-
ARP machine.

ARP is also used to negotiate the use of trailer IP encapsu-
lations; trailers are an alternate encapsulation used to
allow efficient packet alignment for large packets despite
variable-sized headers. Hosts which wish to receive trailer
encapsulations so indicate by sending gratuitous ARP trans-
lation replies along with replies to IP requests; they are
also sent in reply to IP translation replies. The negotia-
tion is thus fully symmetrical, in that either or both hosts
may request trailers. The ATF_USETRAILERS flag is used to
record the receipt of such a reply, and enables the
transmission of trailer packets to that host.

ARP watches passively for hosts impersonating the local host
(i.e. a host which responds to an ARP mapping request for
the local host's address).

DIAGNOSTICS
     duplicate IP address!! sent from ethernet address:
     %x:%x:%x:%x:%x:%x. ARP has discovered another host on the
     local network which responds to mapping requests for its own
     Internet address.

SEE ALSO
     ec(4), de(4), il(4), inet(4F), arp(8C), ifconfig(8C)
     ``An Ethernet Address Resolution Protocol,'' RFC826, Dave
     Plummer, Network Information Center, SRI.
     ``Trailer Encapsulations,'' RFC893, S.J. Leffler and M.J.
     Karels, Network Information Center, SRI.

BUGS
     ARP packets on the Ethernet use only 42 bytes of data; how-
     ever, the smallest legal Ethernet packet is 60 bytes (not
     including CRC). Some systems may not enforce the minimum
     packet size, others will.

Printed 6/17/012 August 1, 1987   2

NAME
     bk - line discipline for machine-machine communication
     (obsolete)

SYNOPSIS
     Not currently supported under 2.11BSD

DESCRIPTION
     This line discipline provides a replacement for the old and
     new tty drivers described in tty(4) when high speed output
     to and especially input from another machine is to be
     transmitted over a asynchronous communications line. The
     discipline was designed for use by the Berkeley network. It
     may be suitable for uploading of data from microprocessors
     into the system. If you are going to send data over asyn-
     chronous communications lines at high speed into the system,
     you must use this discipline, as the system otherwise may
     detect high input data rates on terminal lines and disables
     the lines; in any case the processing of such data when nor-
     mal terminal mechanisms are involved saturates the system.

     The line discipline is enabled by a sequence:

        #include <sgtty.h>
        int ldisc = NETLDISC, fildes; ...
        ioctl(fildes, TIOCSETD, &ldisc);

     A typical application program then reads a sequence of lines
     from the terminal port, checking header and sequencing
     information on each line and acknowledging receipt of each
     line to the sender, who then transmits another line of data.
     Typically several hundred bytes of data and a smaller amount
     of control information will be received on each handshake.

     The old standard teletype discipline can be restored by
     doing:

        ldisc = OTTYDISC;
        ioctl(fildes, TIOCSETD, &ldisc);

     While in networked mode, normal teletype output functions
     take place. Thus, if an 8 bit output data path is desired,
     it is necessary to prepare the output line by putting it
     into RAW mode using ioctl(2). This must be done before
     changing the discipline with TIOCSETD, as most ioctl(2)
     calls are disabled while in network line-discipline mode.

     When in network mode, input processing is very limited to
     reduce overhead. Currently the input path is only 7 bits
     wide, with newline the only recognized character, terminat-
     ing an input record. Each input record must be read and
     acknowledged before the next input is read as the system

Printed 6/17/012 January 27, 1996   1

refuses to accept any new data when there is a record in the
buffer. The buffer is limited in length, but the system
guarantees to always be willing to accept input resulting in
512 data characters and then the terminating newline.

User level programs should provide sequencing and checksums
on the information to guarantee accurate data transfer.

SEE ALSO
    tty(4)

DIAGNOSTICS
    None.

BUGS
    The Purdue uploading line discipline, which provides 8 bits
    and uses timeout's to terminate uploading should be incor-
    porated into the standard system, as it is much more suit-
    able for microprocessor connections.

NAME
     br - EATON BR1537/BR1711 1538[A,B,C,D] moving head disk

SYNOPSIS
     /sys/conf/SYSTEM:
        NBR br_drives # EATON 1537/1711, EATON 1538A, B, C, D

     /etc/dtab:
        #Name Unit# Addr   Vector Br Handler(s)   # Comments
        br   ? 176710   254   5 brintr   # Eaton 1537/8

     major device number(s):
        raw: 20
        block: 11
     minor device encoding:
        bits 0007 specify partition of BR drive
        bits 0070 specify BR drive

DESCRIPTION
     Files with minor device numbers 0 through 7 refer to various
     portions of drive 0; minor devices 8 through 15 refer to
     drive 1, etc. There are four drive types supported by the
     Eaton BR1537 and BR1711 controllers, these are the 1538A (50
     Mb), 1538B (80 Mb), 1538C (200 Mb) and 1538D (300 Mb).
     Capacities are unformatted megabytes. The standard device
     names begin with ``br'' followed by the drive number and
     then a letter a-h for partitions 0-7 respectively. The
     character ? stands here for a drive number in the range 0-7.

     The block files access the disk via the system's normal
     buffering mechanism and may be read and written without
     regard to physical disk records. There is also a `raw'
     interface which provides for direct transmission between the
     disk and the user's read or write buffer.   A single read or
     write call results in exactly one I/O operation and there-
     fore raw I/O is considerably more efficient when many words
     are transmitted. The names of the raw files conventionally
     begin with an extra `r.'

     In raw I/O the buffer must begin on a word (even) boundary,
     and counts should be a multiple of 512 bytes (a disk sec-
     tor). Likewise seek calls should specify a multiple of 512
     bytes.

DISK SUPPORT
     The size (in sectors) of the pseudo-disks on each drive are
     as follows:

```
     1538A partitions:
       Disk      length    cyls       comments
       br?a      18260      0 - 165   1538A has 22 sec/trk, 5 trk/cyl
       br?b      12210    166 - 276
       br?c      59180    277 - 814
       br?d      unused
       br?e      unused
       br?f      unused
       br?g      unused
       br?h      89650      0 - 814

     1538B partitions:
       Disk      length    cyls       comments
       br?a      18400      0 - 114   1538B has 32 sec/trk, 5 trk/cyl
       br?b      12320    115 - 190
       br?c      99840    191 - 814
       br?d      unused
       br?e      unused
       br?f      unused
       br?g      unused
       br?h      99840      0 - 814

     1538C partitions:
       Disk      length    cyls       comments
       br?a      18392      0 -  43   1538C has 22 sec/trk, 19 trk/cyl
       br?b      12122     44 -  72
       br?c     231990     73 - 627
       br?d      78166    628 - 814
       br?e      unused
       br?f      unused
       br?g      unused
       br?h     340670      0 - 814

     1538D partitions:
       Disk      length    cyls       comments
       br?a      18240      0 -  29   1538D has 32 sec/trk, 19 trk/cyl
       br?b      12160     30 -  49
       br?c     232256     50 - 431
       br?d     232256    432 - 813
       br?e      unused
       br?f      unused
       br?g      unused
       br?h     495520      0 - 814
```

FILES
```
    /dev/br[0-7][a-h]      block files
    /dev/rbr[0-7][a-h]     raw files
    /dev/MAKEDEV           script to create special files
    /dev/MAKEDEV.local     script to localize special files
```

SEE ALSO
     ra(4), ram(4), rk(4), rl(4), rx(4), si(4), xp(4), dtab(5),

autoconfig(8)

DIAGNOSTICS
     br%d%c: hard error sn%d cs2=%b ds=%b er=%b. An unrecover-
     able error occurred during transfer of the specified sector
     of the specified disk partition. The contents of the cs2,
     ds and er registers are printed in octal and symbolically
     with bits decoded. The error was either unrecoverable, or a
     large number of retry attempts (including offset positioning
     and drive recalibration) could not recover the error.

BUGS
     In raw I/O read and write(2) truncate file offsets to 512-
     byte block boundaries, and write scribbles on the tail of
     incomplete blocks. Thus, in programs that are likely to
     access raw devices, read, write and lseek(2) should always
     deal in 512-byte multiples.

     DEC-standard error logging should be supported.

     A program to analyze the logged error information (even in
     its present reduced form) is needed.

     The partition tables for the file systems should be read off
     of each pack, as they are never quite what any single
     installation would prefer, and this would make packs more
     portable.

     Only the 1538D (300Mb) disk has been used with this driver,
     the disktab(5) file and newfs(8) program only know about the
     1538D.

NAME
     cons - PDP-11 console interface

DESCRIPTION
     The console is available to the processor through the con-
     sole registers. It acts like a normal terminal, except that
     typing control-P on the consoles of some PDP-11's puts the
     console in local console mode with an ODT (console debugger
     mode) prompt. The operation of the console in this mode
     varies per-processor. See the PDP-11 Processor Handbook for
     your particular processor.

     On an 11/44 the console can be put into local mode by typing
     a control-P. The ODT prompt is ``>>>''. The processor is
     not stopped by entering local console mode. The CPU may be
     halted with the ``halt'' command, which may be abbreviated
     to ``h.'' Conversational mode is re-entered by using the
     command ``c'' (continue).

     On 11/73 the processor's front panel halt switch must be
     pressed to put the console into ODT mode.   The ODT prompt is
     an ``@''.   No characters typed on the console are special
     while UNIX is in control of the console.

FILES
     /dev/console

SEE ALSO
     tty(4), reboot(8)
     PDP-11 Hardware Handbook specific to hardware

NAME
     css - DEC IMP-11A LH/DH IMP interface

SYNOPSIS
     /sys/conf/SYSTEM:
        NCSS css_controllers   # DEC/CSS IMP11-A ARPAnet IMP interface

DESCRIPTION
     The css device provides a Local Host/Distant Host interface
     to an IMP. It is normally used when participating in the
     DARPA Internet. The controller itself is not accessible to
     users, but instead provides the hardware support to the IMP
     interface described in imp(4). When configuring, the
     imp(NIMP) pseudo-device is also included.

DIAGNOSTICS
     css%d: not alive. The initialization routine was entered
     even though the device did not autoconfigure. This is indi-
     cates a system problem.

     css%d: can't initialize. Insufficient UNIBUS resources
     existed to initialize the device. This is likely to occur
     when the device is run on a buffered data path on an 11/750
     and other network interfaces are also configured to use buf-
     fered data paths, or when it is configured to use buffered
     data paths on an 11/730 (which has none).

     css%d: imp doesn't respond, icsr=%b. The driver attempted
     to initialize the device, but the IMP failed to respond
     after 500 tries. Check the cabling.

     css%d: stray output interrupt csr=%b. An interrupt occurred
     when no output had previously been started.

     css%d: output error, ocsr=%b icsr=%b. The device indicated
     a problem sending data on output.

     css%d: recv error, csr=%b. The device indicated a problem
     receiving data on input.

     css%d: bad length=%d. An input operation resulted in a data
     transfer of less than 0 or more than 1008 bytes of data into
     memory (according to the word count register). This should
     never happen as the maximum size of a host-IMP message is
     1008 bytes.

NAME
     de - DEC DEUNA 10 Mb/s Ethernet interface

SYNOPSIS
     /sys/conf/SYSTEM:
        NDE   de_controllers # DEUNA

DESCRIPTION
     The de interface provides access to a 10 Mb/s Ethernet net-
     work through a Digital Equipment UNIBUS Network Adapter
     (DEUNA).

     Each of the host's network addresses is specified at boot
     time with an SIOCSIFADDR ioctl. The de interface employs
     the address resolution protocol described in arp(4P) to
     dynamically map between Internet and Ethernet addresses on
     the local network.

     The interface normally tries to use a ``trailer'' encapsula-
     tion to minimize copying data on input and output. The use
     of trailers is negotiated with ARP. This negotiation may be
     disabled, on a per-interface basis, by setting the
     IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

DIAGNOSTICS
     de%d: hardware address %s. This is a normal autoconfigura-
     tion message noting the 6 byte physical ethernet address of
     the adapter.

     de%d: oerror, flags=%b tdrerr=%b (len=%d). The hardware
     indicated an error in transmitting a packet to the cable.
     The status and error flags are reported.

     de%d: ierror, flags=%b lenerr=%b (len=%d). The hardware
     indicated an error in reading a packet from the cable. The
     status and error flags are reported.

     de%d: can't handle af%d. The interface was handed a message
     with addresses formatted in an unsuitable address family;
     the packet was dropped.

     de%d: buffer unavailable.   The interface received more pack-
     ets than it had buffers allocated to receive them.

     de%d: address change failed, csr0=%b csr1=%b. The interface
     was unable to reprogram its physical ethernet address. This
     may happen with very early models of the interface. This
     facility is used only when the controller is not the first
     network interface configured for XNS.

     The following messages indicate a probable hardware error
     performing the indicated operation during autoconfiguration

Printed 6/17/012 August 20, 1987   1

or initialization. The two control and status registers
should indicate the nature of the failure. See the hardware
manual for details.

de%d: reset failed, csr0=%b csr1=%b.

de%d: ppcb failed, csr0=%b csr1=%b.

de%d: read addr failed, csr0=%b csr1=%b.

de%d: wtring failed, csr0=%b csr1=%b.

de%d: wtmode failed, csr0=%b csr1=%b.

SEE ALSO
     intro(4N), inet(4F), arp(4P)

NAME
     dh - DH-11/DM-11 communications multiplexer

SYNOPSIS
     /sys/conf/SYSTEM:
        NDH dh_units # DH11; NDH is in units of boards (16 each)
        NDM dm_units # DM11; NDM is in units of boards (16 each)

     /etc/dtab:
        #Name   Unit# Addr Vector Br Handler(s)   # Comments
        dm ?   170500 310   4 dmintr   # dm11 dh modem control
        dh ?   160020 320   5 dhrint dhxint   # dh11 terminal mux

     major device number(s):
        raw: 3
     minor device encoding:
        bits 0017 specify line on DH unit
        bits 0060 specify DH unit
        bit 0100 specify RTS/CTS (``hardware'') flowcontrol
        bit 0200 specifies non-blocking open (``CD always on'')

DESCRIPTION
     A dh-11 provides 16 communication lines; dm-11's may be
     optionally paired with dh-11's to provide modem control for
     the lines. If there are fewer dm-11's than dh-11's, the
     dm-11's are assumed to be paired with the first dm_units
     dh-11's.

     Each line attached to the DH-11 communications multiplexer
     behaves as described in tty(4). Input and output for each
     line may independently be set to run at any of 16 speeds;
     see tty(4) for the encoding.

     Bit 0200 of the minor device number for DH lines may be set
     to say that a line is not properly connected, and that the
     line should be treated as hard-wired with carrier always
     present. Thus creating the special character device node
     "3, 130" via mknod /dev/ttyh2 c 3 130 would cause line ttyh2
     to be treated in this way. Bit 0100 of the minor device
     number enables RTS/CTS (also called ``hardware'') flow con-
     trol. It is enabled by adding 64 to the minor device number:
     mknod /dev/ttyh2 c 3 194

     The dh driver monitors the rate of input on each board, and
     switches between the use of character-at-a-time interrupts
     and input silos. While the silo is enabled during periods
     of high-speed input, the driver polls for input 30 times per
     second.

FILES
     /dev/tty[h-k][0-9a-f]
     /dev/ttyd[0-9a-f]

        /dev/MAKEDEV   script to create special files
        /dev/MAKEDEV.local   script to localize special files

SEE ALSO
        tty(4), dtab(5), autoconfig(8)

DIAGNOSTICS
        dh%d NXM.   No response from UNIBUS on a dma transfer within
        a timeout period. This has never been observed on a PDP-11
        and is a carryover from the VAX driver when it was ported.
        It is not serious.

        dh%d %d overruns. The character input silo overflowed
        before it could be serviced. This message is only printed
        at line close time. It is not serious but does indicate
        that the system was not able to keep up with the data flow.

NAME
     dhu - DHU-11 communications multiplexer

SYNOPSIS
     /sys/conf/SYSTEM:
        NDHU dhu_units # DHU11

     /etc/dtab:
        #Name Unit# Addr   Vector Br Handler(s)   # Comments
        du ? 160020 310   5 dhurint dhuxint   # dhu11 terminal mux

     major device number(s):
        raw: 4
     minor device encoding:
        bits 0017 specify line on DHU unit
        bits 0060 specify DHU unit
        bit 0100 specify RTS/CTS flow control
        bit 0200 specifies non-blocking open (``CD always on'')

DESCRIPTION
     A DHU-11 provides 16 communication lines.

     Each line attached to the DHU-11 communications multiplexer
     behaves as described in tty(4). Input and output for each
     line may independently be set to run at any of 13 speeds
     (50, 200 and 38400 baud are not available); see tty(4) for
     the encoding.

     Bit 0200 of the minor device number for DHU lines may be set
     to say that a line is not properly connected, and that the
     line should be treated as hard-wired with carrier always
     present. Thus creating the special character device node
     "4, 130" via mknod /dev/ttyS2 c 4 130 would cause line ttyS2
     to be treated in this way. Turning on bit 6 (adding 64) to
     the minor device number via mknod /dev/ttyS2 c 4 194 enables
     RTS/CTS flow control.

     The DHU-11 driver normally uses input silos and delays
     receiver interrupts by 20 milliseconds rather than taking an
     interrupt on each input character.

FILES
     /dev/tty[S-Z][0-9a-f]

SEE ALSO
     tty(4)

DIAGNOSTICS
     dhu(%d,%d) NXM. No response from UNIBUS on a DMA transfer
     within a timeout period. This has never been observed on a
     PDP-11 and is believed to be a carryover from the VAX driver
     when it was ported. It is not serious.

Printed 6/17/012   May 31, 1997   1

dhu%d %d overruns. The character input silo overflowed
before it could be serviced. This message is printed only
at line close time rather than on each overrun error. Ker-
nel printf's are not interrupt driven and caused more over-
runs by blocking interrupts for lengthy periods of time.

NOTES
    The driver currently does not make full use of the hardware
    capabilities of the DHU-11, for dealing with XON/XOFF flow-
    control or hard-wired lines for example.

    The 4 40-way cables are a pain.

NAME
     dhv - DHV-11 communications multiplexer

SYNOPSIS
     /sys/conf/SYSTEM:
        NDHV dhv_units # DHV11

     /etc/dtab:
        #Name   Unit# Addr   Vector Br Handler(s)   # Comments
        dhv ?   160440 310   5 dhvrint dhvxint   # dhv terminal mux

     major device number(s):
        raw: 24
     minor device encoding:
        bits 0007 specify line on DHV unit
        bits 0070 specify DHV unit
        bit 0100 specifies RTS/CTS (``hardware'') flowcontrol
        bit 0200 specifies non-blocking open (``CD always on'')

DESCRIPTION
     A DHV-11 provides 8 communication lines.

     Each line attached to the DHV-11 communications multiplexer
     behaves as described in tty(4). Input and output for each
     line may independently be set to run at any of 13 speeds (50
     and 200 baud are not available). While 38400 is available
     the underlying hardware is not fast enough to handle it and
     there will be pauses/gaps between characters.

     Bit 0200 of the minor device number for DHV lines may be set
     to say that a line is not properly connected, and that the
     line should be treated as hard-wired with carrier always
     present. Thus creating the special character device node
     "4, 130" via mknod /dev/ttyS2 c 4 130 would cause line ttyS2
     to be treated in this way. Turning on bit 6 in the minor
     device number via mknod /dev/ttyS2 c 4 194 would enable
     RTS/CTS flow control.

     The DHV-11 has an input silo but does not have the programm-
     able receiver delay that the DHU (and DHQ) have. Thus sys-
     tem services more interrupts (i.e. gets fewer characters per
     interrupt on average) with a DHV-11 than with a DHQ (in DHU
     mode).

FILES
     /dev/tty[S-Z][0-9a-f]

SEE ALSO
     tty(4)

DIAGNOSTICS
     dhv%d,%d NXM. No response from QBUS on a DMA transfer

within a timeout period. This error has never been observed
on a PDP-11 and is a carryover from the VAX driver when that
was ported to 2BSD.

dhv%d diag %o. Diagnostic information from the DHV11. This
has never been observed. The DHV-11 hardware manual will be
required to decode the value printed out.

dhv%d: %d overruns. The character input silo overflowed
before it could be serviced. This message is printed only
when the line is closed. By only printing this when the
line is closed further silo overruns are avoided (kernel
printf statements are not interrupt driven).

NOTES
     The DHV lacks the receiver delay that the DHU (and DHQ)
     have. Thus it is extremely easy (indeed it's almost certain
     at higher data rates) for a DHV-11 to enter an interrupt per
     character mode and use 70-80% of the cpu.

NAME
     dmc - DEC DMC-11/DMR-11 point-to-point communications device

SYNOPSIS
     /sys/conf/SYSTEM:
       NDMC dmc_controllers   # DMC11

DESCRIPTION
     The dmc interface provides access to a point-to-point com-
     munications device which runs at either 1 Mb/s or 56 Kb/s.
     DMC-11's communicate using the DEC DDCMP link layer proto-
     col.

     The dmc interface driver also supports a DEC DMR-11 provid-
     ing point-to-point communication running at data rates from
     2.4 Kb/s to 1 Mb/s. DMR-11's are a more recent design and
     thus are preferred over DMC-11's. The NXMT and NRCV con-
     stants in the driver should be increased in this case, as
     the DMR can accept up to 64 transmit and receive buffers, as
     opposed to 7 for the DMC.

     The configuration flags specify how to set up the device,
     0 -- full duplex DDCMP (normal mode)
     1 -- DDCMP Maintence mode (generally useless)
     2 -- DDCMP Half Duplex, primary station
     3 -- DDCMP Half Duplex, secondary station

     Several device error counters are available via "adb", for
     more information see the adb script /usr/share/adb/dmcstats,
     or the DMC11 technical manual.

     The host's address must be specified with an SIOCSIFADDR
     ioctl, and the destination address specified with a SIOC-
     SIFDSTADDR ioctl, before the interface will transmit or
     receive any packets.

ROUTING
     The driver places a HOST entry in the kernel routing tables
     for the address given in the SIOCSIFDSTADDR ioctl. To use
     the DMC as a link between local nets, the route to the
     remote net must be added manually with the route(8) command,
     or by the use of the routing process routed(8) on each end
     of the link.

DIAGNOSTICS
     dmc%d: bad control %o. A bad parameter was passed to the
     dmcload routine.

     dmc%d: unknown address type %d. An input packet was
     received which contained a type of address unknown to the
     driver.

DMC fatal error 0%o. A fatal error in DDMCP occurred, caus-
ing the device to be restarted.

DMC soft error 0%o. A non-fatal error in DDMCP has
occurred.

dmc%d: af%d not supported. The interface was handed a mes-
sage which has addresses formatted in an unsuitable address
family.

SEE ALSO
     intro(4N), inet(4F)

BUGS
     The current version of the driver uses a link-level encapsu-
     lation so that multiple protocol types may be used. It is
     thus incompatible with earlier drivers, including the 4.2BSD
     version.

NAME
     dr - DR11-W general purpose interface driver

SYNOPSIS
     /sys/conf/SYSTEM:
        NDR dr11-w_units   # DR11-W

     /etc/dtab:
        #Name   Unit# Addr   Vector Br Handler(s) # Comments
        dr ?   172410 124   5 drintr   # dr11-w (b-bus emulator)
        dr ?   172430 130   5 drintr   # dr11-w (gc-bus emulator)
        dr ?   172450 134   5 drintr   # dr11-w (em-2 interface)

     major device number(s):
        raw: 12
     minor device encoding:
        bits 0007 specify DR unit

DESCRIPTION
     The Digital Equipment 1981-82 Peripherals Handbook states
     the following about the DR11-W:

        The The DR11-W is a general purpose, direct memory
        access (DMA) interface to the PDP-11 UNIBUS or VAX
        UNIBUS. The DR11-W moves data directly between memory
        and the UNIBUS to and from the user's peripheral.

        It features:

        +   Word or byte transfers.

        +   Programmed or direct memory access (DMA) block
             transfers.

        +   Burst data transfers.

        +   User-controlled transfer rates up to memory speed.

        The DR11-W is a 53-line direct memory access (DMA)
        interface to the PDP-11 UNIBUS or VAX UBA, which allows
        the user to control data transfers between the host
        processor and a peripheral. The DR11-W has 32 data
        lines (for transferring 16-bit parallel data) and 21
        control lines (for transferring control and status
        information.

        When used as an interprocessor buffer (IPB), the DR11-W
        allows data transfers between two processors. Inter-
        processor communication is accomplished by attaching
        one DR11-W to each processor UNIBUS or UBA and then
        cabling the two DR11-Ws together.

Printed 6/17/012 January 27, 1996   1

There. Now you know as much about it as I do.

A number of ioctl(2) calls apply to the dr devices, and have
the form
   #include <pdpuba/drreg.h>
   ioctl(fildes, code, arg)
   int *arg;
The applicable codes are:

DRGTTY   Get DR11-W status.

DRSTTY   Set flags and function.

DRSFUN   Set function.

DRSFLAG   Set flags.

DRGCSR   Get CSR and WCR.

DRSSIG   Set signal for ATTN interrupt.

DRESET   Reset DR11-W interface.

DRSTIME   Set timeout.

DRCTIME   Set timeout inactive.

DROUTPUT   Put word in output data register.

DRINPUT   Get word from input data register.

DRITIME   Don't ignore errors on timeout.

FILES
     /dev/dr[0-7] device special files
     /dev/MAKEDEV script to create special files
     /dev/MAKEDEV.local script to localize special files

SEE ALSO
     dtab(5), autoconfig(8)

DIAGNOSTICS
     dr%d: error csr=%b, eir=%b

     dr%d: timeout error

BUGS
     This interface is only available under 2.9BSD and 2.11BSD.
     No documentation exists on how to use it.

DZ(4)   UNIX Programmer's Manual   DZ(4)

NAME
     dz - DZ-11 communications multiplexer

SYNOPSIS
     /sys/conf/SYSTEM:
         NDZ dz_units # DZ11; NDZ is in units of boards (8 each)

     /etc/dtab:
         #Name   Unit# Addr   Vector Br Handler(s) # Comments
         dz ?   160100 310   5 dzrint dzdma   # dz11 terminal mux

     major device number:
         raw: 2
     minor device encoding:
         bits 0007 specify line on DZ unit
         bits 0170 specify DZ unit
         bit 0200 specifies non-blocking open (``CD always on'')

DESCRIPTION
     A DZ11 provides 8 communication lines with partial modem
     control, adequate for UNIX dialup use. Each line attached
     to the DZ11 communications multiplexer behaves as described
     in tty(4) and may be set to run at any of 16 speeds; see
     tty(4) for the encoding.

     Bit 0200 of the minor device number for DZ lines may be set
     to say that a line is not properly connected, and that the
     line should be treated as hard-wired with carrier always
     present. Thus creating the special character device node
     "2, 130" via mknod /dev/tty02 c 2 130 would cause line tty02
     to be treated in this way.

     The dz driver monitors the rate of input on each board, and
     switches between the use of character-at-a-time interrupts
     and input silos. While the silo is enabled during periods
     of high-speed input, the driver polls for input 30 times per
     second.

FILES
     /dev/tty[0-9][0-9]
     /dev/ttyd[0-9a-f]   dialups
     /dev/MAKEDEV   script to create special files
     /dev/MAKEDEV.local   script to localize special files

SEE ALSO
     tty(4), dtab(5), autoconfig(8)

DIAGNOSTICS
     dz%d: silo overflow. The 64 character input silo overflowed
     before it could be serviced. This can happen if a hard
     error occurs when the CPU is running with elevated priority,
     as the system will then print a message on the console with

Printed 6/17/012 January 28, 1988   1

interrupts disabled. It is not serious.

NAME
     ec - 3Com 10 Mb/s Ethernet interface

SYNOPSIS
     /sys/conf/SYSTEM:
         NEC ec_controllers # 3Com Ethernet

DESCRIPTION
     The ec interface provides access to a 10 Mb/s Ethernet net-
     work through a 3com controller.

     The hardware has 32 kilobytes of dual-ported memory on the
     UNIBUS. This memory is used for internal buffering by the
     board, and the interface code reads the buffer contents
     directly through the UNIBUS. The address of this memory is
     given in the flags field in the configuration file. The
     first interface normally has its memory at Unibus address 0.

     Each of the host's network addresses is specified at boot
     time with an SIOCSIFADDR ioctl. The ec interface employs
     the address resolution protocol described in arp(4P) to
     dynamically map between Internet and Ethernet addresses on
     the local network.

     The interface normally tries to use a ``trailer'' encapsula-
     tion to minimize copying data on input and output. The use
     of trailers is negotiated with ARP. This negotiation may be
     disabled, on a per-interface basis, by setting the
     IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

     The interface software implements an exponential backoff
     algorithm when notified of a collision on the cable. This
     algorithm utilizes a 16-bit mask and the VAX-11's interval
     timer in calculating a series of random backoff values. The
     algorithm is as follows:

     1.   Initialize the mask to be all 1's.

     2.   If the mask is zero, 16 retries have been made and we
          give up.

     3.   Shift the mask left one bit and formulate a backoff by
          masking the interval timer with the smaller of the com-
          plement of this mask and a 5-bit mask, resulting in a
          pseudo-random number between 0 and 31. This produces
          the number of slot times to delay, where a slot is 51
          microseconds.

     4.   Use the value calculated in step 3 to delay before
          retransmitting the packet. The delay is done in a
          software busy loop.

DIAGNOSTICS
     ec%d: send error. After 16 retransmissions using the
     exponential backoff algorithm described above, the packet
     was dropped.

     ec%d: input error (offset=%d). The hardware indicated an
     error in reading a packet off the cable or an illegally
     sized packet. The buffer offset value is printed for debug-
     ging purposes.

     ec%d: can't handle af%d. The interface was handed a message
     with addresses formatted in an unsuitable address family;
     the packet was dropped.

SEE ALSO
     intro(4N), inet(4F), arp(4P)

BUGS
     The hardware is not capable of talking to itself.   The
     software implements local sending and broadcast by sending
     such packets to the loop interface. This is a kludge.

     Backoff delays are done in a software busy loop. This can
     degrade the system if the network experiences frequent col-
     lisions.

NAME
     en - Xerox 3 Mb/s Ethernet interface

SYNOPSIS
     /sys/conf/SYSTEM:
        NEN   en_controllers # Xerox prototype (3 Mb) Ethernet

DESCRIPTION
     The en interface provides access to a 3 Mb/s Ethernet net-
     work. Due to limitations in the hardware, DMA transfers to
     and from the network must take place in the lower 64K bytes
     of the UNIBUS address space, and thus this must be among the
     first UNIBUS devices enabled after boot.

     Each of the host's network addresses is specified at boot
     time with an SIOCSIFADDR ioctl. The station address is
     discovered by probing the on-board Ethernet address regis-
     ter, and is used to verify the protocol addresses. No pack-
     ets will be sent or accepted until a network address is sup-
     plied.

     The interface software implements an exponential backoff
     algorithm when notified of a collision on the cable. This
     algorithm utilizes a 16-bit mask and the VAX-11's interval
     timer in calculating a series of random backoff values. The
     algorithm is as follows:

     1.   Initialize the mask to be all 1's.

     2.   If the mask is zero, 16 retries have been made and we
          give up.

     3.   Shift the mask left one bit and formulate a backoff by
          masking the interval timer with the mask (this is actu-
          ally the two's complement of the value).

     4.   Use the value calculated in step 3 to delay before
          retransmitting the packet.

     The interface handles both Internet and NS protocol fami-
     lies. It normally tries to use a ``trailer'' encapsulation
     to minimize copying data on input and output. The use of
     trailers is negotiated with ARP. This negotiation may be
     disabled, on a per-interface basis, by setting the
     IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

DIAGNOSTICS
     en%d: output error. The hardware indicated an error on the
     previous transmission.

     en%d: send error. After 16 retransmissions using the
     exponential backoff algorithm described above, the packet

Printed 6/17/012 August 20, 1987   1

was dropped.

en%d: input error. The hardware indicated an error in read-
ing a packet off the cable.

en%d: can't handle af%d. The interface was handed a message
with addresses formatted in an unsuitable address family;
the packet was dropped.

SEE ALSO
    intro(4N), inet(4F)

BUGS
    The device has insufficient buffering to handle back to back
    packets. This makes use in a production environment pain-
    ful.

    The hardware does word at a time DMA without byte swapping.
    To compensate, byte swapping of user data must either be
    done by the user or by the system. A kludge to byte swap
    only IP packets is provided if the ENF_SWABIPS flag is
    defined in the driver and set at boot time with an SIOCSIF-
    FLAGS ioctl.

NAME
     fd, stdin, stdout, stderr file descriptor files

DESCRIPTION
     The files /dev/fd/0 through /dev/fd/# refer to file descrip-
     tors which can be accessed through the file system. If the
     file descriptor is open and the mode the file is being
     opened with is a subset of the mode of the existing descrip-
     tor, the call:

          fd = open("/dev/fd/0", mode);

     and the call:

          fd = fcntl(0, F_DUPFD, 0);

     are equivalent.

     Opening the files /dev/stdin, /dev/stdout and /dev/stderr is
     equivalent to the following calls:

          fd = fcntl(STDIN_FILENO, F_DUPFD, 0);
          fd = fcntl(STDOUT_FILENO, F_DUPFD, 0);
          fd = fcntl(STDERR_FILENO, F_DUPFD, 0);

     Flags to the open(2) call other than O_RDONLY, O_WRONLY and
     O_RDWR are ignored.

FILES
     /dev/fd/#
     /dev/stdin
     /dev/stdout
     /dev/stderr

SEE ALSO
     tty(4)

NAME
     hk - RK6-11/RK06 and RK07 moving head disk

SYNOPSIS
     /sys/conf/SYSTEM:
        NHK   hk_drives # RK611, RK06/07

     /etc/dtab:
        #Name   Unit# Addr   Vector Br Handler(s)   # Comments
        hk ?   177440 210   5   hkintr   # rk611/711 rk06/07

     major device number(s):
        raw: 13
        block: 4
     minor device encoding:
        bits 0007 specify partition of HK drive
        bits 0070 specify HK drive

DESCRIPTION
     Files with minor device numbers 0 through 7 refer to various
     portions of drive 0; minor devices 8 through 15 refer to
     drive 1, etc. The standard device names begin with ``hk''
     followed by the drive number and then a letter a-h for par-
     titions 0-7 respectively.   The character ? stands here for a
     drive number in the range 0-7.

     The block files access the disk via the system's normal
     buffering mechanism and may be read and written without
     regard to physical disk records. There is also a `raw'
     interface which provides for direct transmission between the
     disk and the user's read or write buffer.   A single read or
     write call results in exactly one I/O operation and there-
     fore raw I/O is considerably more efficient when many words
     are transmitted. The names of the raw files conventionally
     begin with an extra `r.'

     In raw I/O the buffer must begin on a word (even) boundary,
     and counts should be a multiple of 512 bytes (a disk sec-
     tor). Likewise seek calls should specify a multiple of 512
     bytes.

DISK SUPPORT
     Disks must be labeled using either the standalone disklabel
     program on the boot tape or with the disklabel(8) program.
     There are no partition tables coded into the hk drivers,
     these must be placed on the drive with disklabel.

     Traditionally the hk?a partition is used for the root
     filesystem, the b partition as a swap area and the c parti-
     tion for disk to disk copying (it spans the entire disk).

The kernel uses the c partition to access the bad block
information stored at the end of some packs. Extreme care
must be taken when creating file systems on this partition
to avoid overwriting any bad block information present.
User data should use the h partition which should be at
least one track (one cylinder might be best) shorter than
the c partition.

FILES
     /dev/hk[0-7][a-h]   block files
     /dev/rhk[0-7][a-h]   raw files
     /dev/MAKEDEV   script to create special files
     /dev/MAKEDEV.local   script to localize special files

SEE ALSO
     ra(4) ram(4), rk(4), rl(4), rx(4), si(4), xp(4), dtab(5),
     autoconfig(8), disklabel(8)

DIAGNOSTICS
     hk%d%c: hard error sn%d cs2=%b ds=%b er=%b. An unrecover-
     able error occurred during transfer of the specified sector
     of the specified disk partition. The contents of the cs2,
     ds and er registers are printed in octal and symbolically
     with bits decoded. The error was either unrecoverable, or a
     large number of retry attempts (including offset positioning
     and drive recalibration) could not recover the error.

     hk%d: write locked. The write protect switch was set on the
     drive when a write was attempted. The write operation is
     not recoverable.

     hk%d: not ready. The drive was spun down or off line when
     it was accessed. The i/o operation is not recoverable.

     hk%d: not ready (came back!). The drive was not ready, but
     after printing the message about being not ready (which
     takes a fraction of a second) was ready. The operation is
     recovered if no further errors occur.

     hk%d%c: soft ecc sn%d. A recoverable ECC error occurred on
     the specified sector of the specified disk partition. This
     happens normally a few times a week. If it happens more
     frequently than this the sectors where the errors are occur-
     ing should be checked to see if certain cylinders on the
     pack, spots on the carriage of the drive or heads are indi-
     cated.

BUGS
     In raw I/O read and write(2) truncate file offsets to 512-
     byte block boundaries, and write scribbles on the tail of
     incomplete blocks. Thus, in programs that are likely to
     access raw devices, read, write and lseek(2) should always

Printed 6/17/012 December 28, 1997   2

deal in 512-byte multiples.

DEC-standard error logging should be supported.

A program to analyze the logged error information (even in its present reduced form) is needed.

NAME
     ht - TM-03/TE-16,TU-45,TU-77 MASSBUS magtape interface

SYNOPSIS
     /sys/conf/SYSTEM:
       NHT   ht_drives # TE16, TU45, TU77

     /etc/dtab:
       #Name   Unit# Addr   Vector Br Handler(s)   # Comments
       ht ?   172440 224   5 htintr   # tu 16 massbus tape

     major device number(s):
       raw: 6
       block: 0
     minor device encoding:
       bits 0003 specify HT drive
       bit 0004 specifies no-rewind operation
       bit 0010 specifies 1600BPI recording density instead of 800BPI

DESCRIPTION
     The tm-03/transport combination provides a standard tape
     drive interface as described in mtio(4). All drives provide
     both 800 and 1600 bpi; the TE-16 runs at 45 ips, the TU-45
     at 75 ips, while the TU-77 runs at 125 ips and autoloads
     tapes.

FILES
     /dev/MAKEDEV   script to create special files
     /dev/MAKEDEV.local   script to localize special files

SEE ALSO
     mt(1), tar(1), tp(1), mtio(4), tm(4), ts(4), dtab(5), auto-
     config(8)

DIAGNOSTICS
     tu%d: no write ring. An attempt was made to write on the
     tape drive when no write ring was present; this message is
     written on the terminal of the user who tried to access the
     tape.

     tu%d: not online. An attempt was made to access the tape
     while it was offline; this message is written on the termi-
     nal of the user who tried to access the tape.

     tu%d: can't change density in mid-tape. An attempt was made
     to write on a tape at a different density than is already
     recorded on the tape. This message is written on the termi-
     nal of the user who tried to switch the density.

     tu%d: hard error bn%d er=%b ds=%b.   A tape error occurred
     at block bn; the ht error register and drive status register
     are printed in octal with the bits symbolically decoded.

Printed 6/17/012 January 28, 1988   1

Any error is fatal on non-raw tape; when possible the driver
will have retried the operation which failed several times
before reporting the error.

BUGS

If any non-data error is encountered on non-raw tape, it
refuses to do anything more until closed.

The system should remember which controlling terminal has
the tape drive open and write error messages to that termi-
nal rather than on the console.

NAME
     hy - Network Systems Hyperchannel interface

SYNOPSIS
     /sys/conf/SYSTEM:
        NHY   hy_controllers # Hyperchannel

DESCRIPTION
     The hy interface provides access to a Network Systems Cor-
     poration Hyperchannel Adapter.

     The network to which the interface is attached is specified
     at boot time with an SIOCSIFADDR ioctl. The host's address
     is discovered by reading the adapter status register. The
     interface will not transmit or receive packets until the
     network number is known.

DIAGNOSTICS
     hy%d: unit number 0x%x port %d type %x microcode level 0x%x.
     Identifies the device during autoconfiguration.

     hy%d: can't handle af%d. The interface was handed a message
     with addresses formatted in an unsuitable address family;
     the packet was dropped.

     hy%d: can't initialize. The interface was unable to allo-
     cate UNIBUS resources. This is usually due to having too
     many network devices on an 11/750 where there are only 3
     buffered data paths.

     hy%d: NEX - Non Existent Memory. Non existent memory error
     returned from hardware.

     hy%d: BAR overflow. Bus address register overflow error
     returned from hardware.

     hy%d: Power Off bit set, trying to reset.   Adapter has lost
     power, driver will reset the bit and see if power is still
     out in the adapter.

     hy%d: Power Off Error, network shutdown. Power was really
     off in the adapter, network connections are dropped.
     Software does not shut down the network unless power has
     been off for a while.

     hy%d: RECVD MP > MPSIZE (%d). A message proper was received
     that is too big. Probable a driver bug. Shouldn't happen.

     hy%d: xmit error - len > hy_olen [%d > %d]. Probable driver
     error. Shouldn't happen.

    hy%d: DRIVER BUG - INVALID STATE %d. The driver state
    machine reached a non-existent state. Definite driver bug.

    hy%d: watchdog timer expired. A command in the adapter has
    taken too long to complete. Driver will abort and retry the
    command.

    hy%d: adapter power restored. Software was able to reset
    the power off bit, indicating that the power has been
    restored.

SEE ALSO
    intro(4N), inet(4F)

BUGS
    If the adapter does not respond to the status command issued
    during autoconfigure, the adapter is assumed down. A reboot
    is required to recognize it.

    The adapter power fail interrupt seems to occur sporadically
    when power has, in fact, not failed. The driver will
    believe that power has failed only if it can not reset the
    power fail latch after a ``reasonable'' time interval.
    These seem to appear about 2-4 times a day on some machines.
    There seems to be no correlation with adapter rev level,
    number of ports used etc. and whether a machine will get
    these ``bogus powerfails''. They don't seem to cause any
    real problems so they have been ignored.

NAME
     icmp - Internet Control Message Protocol

SYNOPSIS
     #include <sys/socket.h>
     #include <netinet/in.h>

     s = socket(AF_INET, SOCK_RAW, proto);

DESCRIPTION
     ICMP is the error and control message protocol used by IP
     and the Internet protocol family. It may be accessed
     through a "raw socket" for network monitoring and diagnostic
     functions. The proto parameter to the socket call to create
     an ICMP socket is obtained from getprotobyname(3N). ICMP
     sockets are connectionless, and are normally used with the
     sendto and recvfrom calls, though the connect(2) call may
     also be used to fix the destination for future packets (in
     which case the read(2) or recv(2) and write(2) or send(2)
     system calls may be used).

     Outgoing packets automatically have an IP header prepended
     to them (based on the destination address). Incoming pack-
     ets are received with the IP header and options intact.

DIAGNOSTICS
     A socket operation may fail with one of the following errors
     returned:

     [EISCONN]   when trying to establish a connection on a
                      socket which already has one, or when trying
                      to send a datagram with the destination
                      address specified and the socket is already
                      connected;

     [ENOTCONN]   when trying to send a datagram, but no desti-
                      nation address is specified, and the socket
                      hasn't been connected;

     [ENOBUFS]   when the system runs out of memory for an
                      internal data structure;

     [EADDRNOTAVAIL]
                      when an attempt is made to create a socket
                      with a network address for which no network
                      interface exists.

SEE ALSO
     send(2), recv(2), intro(4N), inet(4F), ip(4P)

NAME
     idp - Xerox Internet Datagram Protocol

SYNOPSIS
     #include <sys/socket.h>
     #include <netns/ns.h>
     #include <netns/idp.h>

     s = socket(AF_NS, SOCK_DGRAM, 0);

DESCRIPTION
     IDP is a simple, unreliable datagram protocol which is used
     to support the SOCK_DGRAM abstraction for the Internet pro-
     tocol family. IDP sockets are connectionless, and are nor-
     mally used with the sendto and recvfrom calls, though the
     connect(2) call may also be used to fix the destination for
     future packets (in which case the recv(2) or read(2) and
     send(2) or write(2) system calls may be used).

     Xerox protocols are built vertically on top of IDP. Thus,
     IDP address formats are identical to those used by SPP.
     Note that the IDP port space is the same as the SPP port
     space (i.e. a IDP port may be "connected" to a SPP port,
     with certain options enabled below). In addition broadcast
     packets may be sent (assuming the underlying network sup-
     ports this) by using a reserved "broadcast address"; this
     address is network interface dependent.

DIAGNOSTICS
     A socket operation may fail with one of the following errors
     returned:

     [EISCONN]   when trying to establish a connection on a
                     socket which already has one, or when trying
                     to send a datagram with the destination
                     address specified and the socket is already
                     connected;

     [ENOTCONN]   when trying to send a datagram, but no desti-
                     nation address is specified, and the socket
                     hasn't been connected;

     [ENOBUFS]   when the system runs out of memory for an
                     internal data structure;

     [EADDRINUSE]   when an attempt is made to create a socket
                     with a port which has already been allocated;

     [EADDRNOTAVAIL]
                     when an attempt is made to create a socket
                     with a network address for which no network
                     interface exists.

SOCKET OPTIONS
     [SO_HEADERS_ON_INPUT]
               When set, the first 30 bytes of any data
               returned from a read or recv from will be the
               initial 30 bytes of the IDP packet, as
               described by
               struct idp {
                   u_short   idp_sum;
                   u_short   idp_len;
                   u_char   idp_tc;
                   u_char   idp_pt;
                   struct ns_addr idp_dna;
                   struct ns_addr idp_sna;
               };
               This allows the user to determine the packet
               type, and whether the packet was a multi-cast
               packet or directed specifically at the local
               host. When requested, gives the current
               state of the option, (NSP_RAWIN or 0).

     [SO_HEADERS_ON_OUTPUT]
               When set, the first 30 bytes of any data sent
               will be the initial 30 bytes of the IDP
               packet. This allows the user to determine
               the packet type, and whether the packet
               should be multi-cast packet or directed
               specifically at the local host. You can also
               misrepresent the sender of the packet. When
               requested, gives the current state of the
               option. (NSP_RAWOUT or 0).

     [SO_DEFAULT_HEADERS]
               The user provides the kernel an IDP header,
               from which it gleans the Packet Type. When
               requested, the kernel will provide an IDP
               header, showing the default packet type, and
               local and foreign addresses, if connected.

     [SO_ALL_PACKETS]
               When set, this option defeats automatic pro-
               cessing of Error packets, and Sequence Proto-
               col packets.

     [SO_SEQNO]   When requested, this returns a sequence
               number which is not likely to be repeated
               until the machine crashes or a very long time
               has passed. It is useful in constructing
               Packet Exchange Protocol packets.

SEE ALSO
     send(2), recv(2), intro(4N), ns(4F)

Printed 6/17/012   July 30, 1985   2

NAME
     il - Interlan NI1010 10 Mb/s Ethernet interface

SYNOPSIS
     /sys/conf/SYSTEM:
        NIL il_controllers # Interlan Ethernet

DESCRIPTION
     The il interface provides access to a 10 Mb/s Ethernet net-
     work through an Interlan 1010 or 1010A controller.

     Each of the host's network addresses is specified at boot
     time with an SIOCSIFADDR ioctl. The il interface employs
     the address resolution protocol described in arp(4P) to
     dynamically map between Internet and Ethernet addresses on
     the local network.

     The interface normally tries to use a ``trailer'' encapsula-
     tion to minimize copying data on input and output. The use
     of trailers is negotiated with ARP. This negotiation may be
     disabled, on a per-interface basis, by setting the
     IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

DIAGNOSTICS
     il%d: input error. The hardware indicated an error in read-
     ing a packet off the cable or an illegally sized packet.

     il%d: can't handle af%d. The interface was handed a message
     with addresses formatted in an unsuitable address family;
     the packet was dropped.

     il%d: setaddr didn't work. The interface was unable to
     reprogram its physical ethernet address. This may happen
     with very early models of the interface. This facility is
     used only when the controller is not the first network
     interface configured for XNS. The oldest interface tested
     (2.7.1.0.1.45) has never failed in this way.

     The following messages indicate a probable hardware error
     performing the indicated operation during autoconfiguration
     or initialization. The status field in the control and
     status register (the low-order four bits) should indicate
     the nature of the failure. See the hardware manual for
     details.

     il%d: reset failed, csr=%b.

     il%d: status failed, csr=%b.

     il%d: hardware diag failed, csr=%b.

        il%d: verifying setaddr, csr=%b.

        il%d: stray xmit interrupt, csr=%b.

        il%d: can't initialize.

SEE ALSO
        intro(4N), inet(4F), arp(4P)

NAME
     imp - IMP raw socket interface

SYNOPSIS
     #include <sys/socket.h>
     #include <netinet/in.h>
     #include <netimp/if_imp.h>

     s = socket(AF_IMPLINK, SOCK_RAW, proto);

DESCRIPTION
     The raw imp socket provides direct access to the imp(4) net-
     work interface. Users send packets through the interface
     using the send(2) calls, and receive packets with the
     recv(2), calls. All outgoing packets must have an 1822 96-
     bit leader on the front. Likewise, packets received by the
     user will have this leader on the front. The 1822 leader
     and the legal values for the various fields are defined in
     the include file <netimp/if_imp.h>. The raw imp interface
     automatically installs the length and destination address in
     the 1822 leader of all outgoing packets; these need not be
     filled in by the user.

     If the protocol selected, proto, is zero, the socket will
     receive all IMP messages except RFNM and incompletes which
     are not input data for a kernel protocol.   If proto is non-
     zero, only messages for the specified link type will be
     received.

DIAGNOSTICS
     An operation on a socket may fail with one of the following
     errors:

     [EISCONN]   when trying to establish a connection on a
                      socket which already has one, or when trying
                      to send a datagram with the destination
                      address specified and the socket is already
                      connected;

     [ENOTCONN]   when trying to send a datagram, but no desti-
                      nation address is specified, and the socket
                      hasn't been connected;

     [ENOBUFS]   when the system runs out of memory for an
                      internal data structure;

     [ENOBUFS]   eight messages to the destination host are
                      outstanding, and another eight are already
                      queued for output;

     [EADDRNOTAVAIL]
                      when an attempt is made to create a socket

with a network address for which no network
interface exists.

SEE ALSO
intro(4N), inet(4F), imp(4)

NAME
     imp - 1822 network interface

SYNOPSIS
     /sys/conf/SYSTEM:
        NIMP count   # ARPAnet IMP 1822 interface

DESCRIPTION
     The imp interface, as described in BBN Report 1822, provides
     access to an intelligent message processor normally used
     when participating in the Department of Defense ARPA net-
     work. The network interface communicates through a device
     controller, usually an ACC LH/DH or HDH or a DEC IMP-11A,
     with the IMP. The interface is "reliable" and "flow-
     controlled" by the host-IMP protocol.

     To configure IMP support, at least one of acc(4), css(4) or
     hdh(4) must be included. The count specifies the total
     number of IMP connections. The network number on which the
     interface resides is specified at boot time using the SIOC-
     SIFADDR ioctl. The host number is discovered through
     receipt of NOOP messages from the IMP.

     The network interface is always in one of four states: up,
     down, initializing, or going down. When the system is
     booted, the interface is marked down. If the hardware con-
     troller is successfully probed, the interface enters the
     initializing state and transmits three NOOP messages to the
     IMP. It then waits for the IMP to respond with two or more
     NOOP messages in reply. When it receives these messages it
     enters the up state. The ``going down'' state is entered
     only when notified by the IMP of an impending shutdown.
     Packets may be sent through the interface only while it is
     in the up state. Outgoing packets are dropped with the
     error ENETDOWN returned to the caller if the interface is in
     any other state.

DIAGNOSTICS
     imp%d: not configured. A hardware interface could not be
     attached during autoconfiguration because too few IMP
     pseudo-devices were configured.

     imp%d: leader error. The IMP reported an error in a leader
     (1822 message header). This causes the interface to be
     reset and any packets queued up for transmission to be
     purged.

     imp%d: going down in 30 seconds.
     imp%d: going down for hardware PM.
     imp%d: going down for reload software.
     imp%d: going down for emergency reset. The Network Control
     Center (NCC) is manipulating the IMP. By convention these

Printed 6/17/012 August 2, 1987   1

messages are reported to all hosts on an IMP.

imp?: host %x, lost %d rfnms. The IMP had messages out-
standing to the host listed, but no RFNM (Request for Next
Message) messages were received from the IMP in 127 seconds.
The software state for that host is reinitialized.

imp%d: interface reset. The host has received an interface
reset message from the IMP.

imp%d: address reset to x%x (%d/%d). The host has received
a NOOP message which caused it to reset its notion of its
current address. The Internet address is printed in hexade-
cimal, with the host and IMP numbers following. This indi-
cates that the address originally set by ifconfig(8) was
incorrect, that the IMP has undergone an identity crisis, or
that communication between the IMP and the host is being
garbled.

imp%d: data error. The IMP noted an error in data transmit-
ted. The host-IMP interface is reset and the host enters
the init state (awaiting NOOP messages).

imp%d: interface reset. The reset process has been com-
pleted.

imp%d: marked down. After receiving a "going down in 30
seconds" message, and waiting 30 seconds, the host has
marked the IMP unavailable. Before packets may be sent to
the IMP again, the IMP must notify the host, through a
series of NOOP messages, that it is back up.

imp%d: can't handle af%d.   The interface was handed a mes-
sage with addresses formatting in an unsuitable address fam-
ily; the packet was dropped.

SEE ALSO
    intro(4N), inet(4F), acc(4), css(4), hdh(4), implog(8),
    implogd(8)

NAME
     inet - Internet protocol family

SYNOPSIS
     #include <sys/types.h>
     #include <netinet/in.h>

DESCRIPTION
     The Internet protocol family is a collection of protocols
     layered atop the Internet Protocol (IP) transport layer, and
     utilizing the Internet address format. The Internet family
     provides protocol support for the SOCK_STREAM, SOCK_DGRAM,
     and SOCK_RAW socket types; the SOCK_RAW interface provides
     access to the IP protocol.

ADDRESSING
     Internet addresses are four byte quantities, stored in net-
     work standard format: layed out as highest to lowest order
     bytes in memory or ``Big Endian'' (the VAX is word and byte
     reversed, or ``Little Endian''; the PDP-11 is byte reversed
     within each word, or ``Middle Endian''). The include file
     <netinet/in.h> defines this address as a discriminated
     union.

     Sockets bound to the Internet protocol family utilize the
     following addressing structure,

     struct sockaddr_in {
         short   sin_family;
         u_short sin_port;
         struct   in_addr sin_addr;
         char   sin_zero[8];
     };

     Sockets may be created with the local address INADDR_ANY to
     effect "wildcard" matching on incoming messages. The address
     in a connect(2) or sendto(2) call may be given as INADDR_ANY
     to mean ``this host.'' The distinguished address
     INADDR_BROADCAST is allowed as a shorthand for the broadcast
     address on the primary network if the first network config-
     ured supports broadcast.

PROTOCOLS
     The Internet protocol family is comprised of the IP tran-
     sport protocol, Internet Control Message Protocol (ICMP),
     Transmission Control Protocol (TCP), and User Datagram Pro-
     tocol (UDP). TCP is used to support the SOCK_STREAM
     abstraction while UDP is used to support the SOCK_DGRAM
     abstraction. A raw interface to IP is available by creating
     an Internet socket of type SOCK_RAW. The ICMP message pro-
     tocol is accessible from a raw socket.

The 32-bit Internet address contains both network and host parts. It is frequency-encoded; the most-significant bit is clear in Class A addresses, in which the high-order 8 bits are the network number. Class B addresses use the high-order 16 bits as the network field, and Class C addresses have a 24-bit network part. Sites with a cluster of local networks and a connection to the DARPA Internet may chose to use a single network number for the cluster; this is done by using subnet addressing. The local (host) portion of the address is further subdivided into subnet and host parts. Within a subnet, each subnet appears to be an individual network; externally, the entire cluster appears to be a single, uniform network requiring only a single routing entry. Subnet addressing is enabled and examined by the following ioctl(2) commands on a datagram socket in the Internet domain; they have the same form as the SIOCIFADDR command (see intro(4N)).

SIOCSIFNETMASK   Set interface network mask. The network
                   mask defines the network part of the
                   address; if it contains more of the
                   address than the address type would
                   indicate, then subnets are in use.

SIOCGIFNETMASK   Get interface network mask.

SEE ALSO
     ioctl(2), socket(2), intro(4N), tcp(4P), udp(4P), ip(4P),
     icmp(4P)

     An Introductory 4.3BSD Interprocess Communication Tutorial
     (PS1:7).

     An Advanced 4.3BSD Interprocess Communication Tutorial
     (PS1:8).

CAVEAT
     The Internet protocol support is subject to change as the
     Internet protocols develop. Users should not depend on
     details of the current implementation, but rather the ser-
     vices exported.

NAME
     ip - Internet Protocol

SYNOPSIS
     #include <sys/socket.h>
     #include <netinet/in.h>

     s = socket(AF_INET, SOCK_RAW, proto);

DESCRIPTION
     IP is the transport layer protocol used by the Internet pro-
     tocol family. Options may be set at the IP level when using
     higher-level protocols that are based on IP (such as TCP and
     UDP). It may also be accessed through a "raw socket" when
     developing new protocols, or special purpose applications.

     A single generic option is supported at the IP level,
     IP_OPTIONS, that may be used to provide IP options to be
     transmitted in the IP header of each outgoing packet.
     Options are set with setsockopt(2) and examined with get-
     sockopt(2). The format of IP options to be sent is that
     specified by the IP protocol specification, with one excep-
     tion: the list of addresses for Source Route options must
     include the first-hop gateway at the beginning of the list
     of gateways. The first-hop gateway address will be
     extracted from the option list and the size adjusted accord-
     ingly before use. IP options may be used with any socket
     type in the Internet family.

     Raw IP sockets are connectionless, and are normally used
     with the sendto and recvfrom calls, though the connect(2)
     call may also be used to fix the destination for future
     packets (in which case the read(2) or recv(2) and write(2)
     or send(2) system calls may be used).

     If proto is 0, the default protocol IPPROTO_RAW is used for
     outgoing packets, and only incoming packets destined for
     that protocol are received. If proto is non-zero, that pro-
     tocol number will be used on outgoing packets and to filter
     incoming packets.

     Outgoing packets automatically have an IP header prepended
     to them (based on the destination address and the protocol
     number the socket is created with). Incoming packets are
     received with IP header and options intact.

DIAGNOSTICS
     A socket operation may fail with one of the following errors
     returned:

     [EISCONN]   when trying to establish a connection on a
                      socket which already has one, or when trying

                  to send a datagram with the destination
                  address specified and the socket is already
                  connected;

    [ENOTCONN]   when trying to send a datagram, but no desti-
                  nation address is specified, and the socket
                  hasn't been connected;

    [ENOBUFS]   when the system runs out of memory for an
                  internal data structure;

    [EADDRNOTAVAIL]
                  when an attempt is made to create a socket
                  with a network address for which no network
                  interface exists.

    The following errors specific to IP may occur when setting
    or getting IP options:

    [EINVAL]   An unknown socket option name was given.

    [EINVAL]   The IP option field was improperly formed; an
                  option field was shorter than the minimum
                  value or longer than the option buffer pro-
                  vided.

SEE ALSO
    getsockopt(2), send(2), recv(2), intro(4N), icmp(4P),
    inet(4F)

NAME
     lp - line printer

SYNOPSIS
     /sys/conf/SYSTEM:
        NLP   lp_printers # Line Printer
        LP_MAXCOL 132   # Maximum number of columns on line printers

     /etc/dtab:
        #Name Unit# Addr   Vector Br Handler(s)   # Comments
        lp ?   177514 200   4 lpintr   # lp-11 line printer

     major device number(s):
        raw: 5
     minor device encoding:
        bit 01 specifies 64-character set (instead of 96-character set)
        bits 06 are reserved
        bits 0370 specify line printer unit: <lp_unit> * 8

DESCRIPTION
     Lp provides the interface to any of the standard DEC line
     printers on an LP-11 parallel interface. When it is opened
     or closed, a suitable number of page ejects is generated.
     Bytes written are printed.

     The unit number of the printer is specified by the minor
     device after removing the low 3 bits, which act as per-
     device parameters. Currently only the lowest of the low
     three bits is interpreted: if it is set, the device is
     treated as having a 64-character set, rather than a full
     96-character set. In the resulting half-ASCII mode, lower
     case letters are turned into upper case and certain charac-
     ters are escaped according to the following table:

        {   (
        }   )
        `   '
        | !
        ~   ^

     The driver correctly interprets carriage returns, back-
     spaces, tabs, and form feeds. Lines longer than the maximum
     page width are truncated.   The page width is specified via
     the LP_MAXCOL definition, set to 132 in the GENERIC distri-
     bution.

FILES
     /dev/lp
     /dev/MAKEDEV script to create special files
     /dev/MAKEDEV.local script to localize special files

SEE ALSO
     lpr(1), dtab(5), autoconfig(8)

DIAGNOSTICS
     None.

BUGS
     Although the driver supports multiple printers this has
     never been tried.

NAME
     mem, kmem - main memory

SYNOPSIS
     major device number(s):
        raw: 1
     minor device encoding:
        mem: 0; kmem: 1; null: 2

DESCRIPTION
     Mem is a special file that is an image of the main memory of
     the computer. It may be used, for example, to examine (and
     even to patch) the system.

     Byte addresses in mem are interpreted as physical memory
     addresses. References to non-existent locations cause
     errors to be returned.

     The file kmem is the same as mem except that kernel virtual
     memory rather than physical memory is accessed. Only kernel
     virtual addresses that are mapped to memory are allowed.
     Examining and patching device registers is likely to lead to
     unexpected results when read-only or write-only bits are
     present.

     On PDP-11s, the I/O page begins at location 0160000 of kmem
     and the per-process data segment for the current process
     begins at 0140000 and is USIZE clicks (64 bytes each) long.

FILES
     /dev/mem
     /dev/kmem
     /dev/MAKEDEV   script to create special files
     /dev/MAKEDEV.local   script to localize special files

BUGS
     On PDP-11's, specifying an odd kernel or user address, or an
     odd transfer count is [generally] slower than using all even
     parameters.

     On machines with ENABLE/34(tm) memory mapping boards the I/O
     page can be accessed only through kmem.

NAME
     mtio - UNIX magtape interface

DESCRIPTION
     The files mt0, ..., mt15 refer to the UNIX magtape drives,
     which may be on the MASSBUS using the TM03 formatter ht(4),
     or on the UNIBUS using either the TM11 or TS11 formatters
     tm(4) or ts(4). The following description applies to any of
     the transport/controller pairs.

     The files mt0, ..., mt7 are 800bpi (or the transport's
     lowest density), mt8, ..., mt15 are 1600bpi (or the
     transport's second density), and mt16, ..., mt23 are 6250bpi
     (or the transport's third density). (But note that only
     1600 bpi is available with the TS11.) The files mt0, ...,
     mt3, mt8, ..., mt11, and mt16, ..., mt19 are rewound when
     closed; the others are not. When a file open for writing is
     closed, two end-of-files are written. If the tape is not to
     be rewound it is positioned with the head between the two
     tapemarks.

     A standard tape consists of a series of 1024 byte records
     terminated by an end-of-file. To the extent possible, the
     system makes it possible, if inefficient, to treat the tape
     like any other file. Seeks have their usual meaning and it
     is possible to read or write a byte at a time. Writing in
     very small units is inadvisable, however, because it uses
     most of the tape in record gaps.

     The mt files discussed above are useful when it is desired
     to access the tape in a way compatible with ordinary files.
     When foreign tapes are to be dealt with, and especially when
     long records are to be read or written, the `raw' interface
     is appropriate. The associated files are named rmt0, ...,
     rmt23, but the same minor-device considerations as for the
     regular files still apply. A number of other ioctl opera-
     tions are available on raw magnetic tape.   The following
     definitions are from <sys/mtio.h>:

     /*
      * Structures and definitions for mag tape io control commands
      */

     /* structure for MTIOCTOP - mag tape op command */
     struct   mtop {
       short   mt_op;   /* operations defined below */
       daddr_t mt_count;   /* how many of them */
     };

     /* operations */
     #define MTWEOF   0   /* write an end-of-file record */
     #define MTFSF   1   /* forward space file */

```
#define MTBSF    2               /* backward space file */
#define MTFSR    3               /* forward space record */
#define MTBSR    4               /* backward space record */
#define MTREW    5               /* rewind */
#define MTOFFL   6               /* rewind and put the drive offline */
#define MTNOP    7               /* no operation, sets status only */
#define MTCACHE  8               /* enable controller cache */
#define MTNOCACHE 9              /* disable controller cache */


/* structure for MTIOCGET - mag tape get status command */

struct   mtget    {
   short    mt_type;  /* type of magtape device */
/* the following two registers are grossly device dependent */
   short    mt_dsreg; /* ``drive status'' register */
   short    mt_erreg; /* ``error'' register */
/* end device-dependent registers */
   short    mt_resid; /* residual count */
/* the following two are not yet implemented */
   daddr_t  mt_fileno;     /* file number of current position */
   daddr_t  mt_blkno; /* block number of current position */
/* end not yet implemented */
};


/*
 * Constants for mt_type byte.  These are the same
 * for other controllers compatible with the types listed.
 */
#define   MT_ISTS       0x01  /* TS-11 */
#define   MT_ISHT       0x02  /* TM03 Massbus: TE16, TU45, TU77 */
#define   MT_ISTM       0x03  /* TM11/TE10 Unibus */
#define   MT_ISMT       0x04  /* TM78/TU78 Massbus */
#define   MT_ISUT       0x05  /* SI TU-45 emulation on Unibus */
#define   MT_ISCPC      0x06  /* SUN */
#define   MT_ISAR       0x07  /* SUN */
#define   MT_ISTMSCP    0x08  /* DEC TMSCP protocol (TU81, TK50) */


/* mag tape io control commands */
#define   MTIOCTOP _IOW(m, 1, struct mtop)  /* do a mag tape op */
#define   MTIOCGET _IOR(m, 2, struct mtget) /* get tape status */
#define MTIOCIEOT  _IO(m, 3)      /* ignore EOT error */
#define MTIOCEEOT  _IO(m, 4)       /* enable EOT error */


#ifndef KERNEL
#define   DEFTAPE  "/dev/rmt12"
#endif
```

Each read or write call reads or writes the next record on
the tape.   In the write case the record has the same length
as the buffer given. During a read, the record size is
passed back as the number of bytes read, provided it is no
greater than the buffer size; if the record is long, an

error is indicated. In raw tape I/O seeks are ignored. A
zero byte count is returned when a tape mark is read, but
another read will fetch the first record of the new tape
file.

The MTCACHE and MTNOCACHE commands do not apply to all tape
drivers. At present only the TMSCP driver supports those
commands and then only for drives such as the TU81+.

FILES
     /dev/mt?
     /dev/rmt?

SEE ALSO
     mt(1), tar(1), tp(1), ht(4), tm(4), ts(4)

BUGS
     The status should be returned in a device independent for-
     mat.

     The special file naming should be redone in a more con-
     sistent and understandable manner.

NAME
     networking - introduction to networking facilities

SYNOPSIS
     #include <sys/socket.h>
     #include <net/route.h>
     #include <net/if.h>

DESCRIPTION
     This section briefly describes the networking facilities
     available in the system. Documentation in this part of sec-
     tion 4 is broken up into three areas: protocol families
     (domains), protocols, and network interfaces. Entries
     describing a protocol family are marked ``4F,'' while
     entries describing protocol use are marked ``4P.'' Hardware
     support for network interfaces are found among the standard
     ``4'' entries.

     All network protocols are associated with a specific proto-
     col family. A protocol family provides basic services to
     the protocol implementation to allow it to function within a
     specific network environment. These services may include
     packet fragmentation and reassembly, routing, addressing,
     and basic transport. A protocol family may support multiple
     methods of addressing, though the current protocol implemen-
     tations do not. A protocol family is normally comprised of
     a number of protocols, one per socket(2) type. It is not
     required that a protocol family support all socket types. A
     protocol family may contain multiple protocols supporting
     the same socket abstraction.

     A protocol supports one of the socket abstractions detailed
     in socket(2). A specific protocol may be accessed either by
     creating a socket of the appropriate type and protocol fam-
     ily, or by requesting the protocol explicitly when creating
     a socket.   Protocols normally accept only one type of
     address format, usually determined by the addressing struc-
     ture inherent in the design of the protocol family/network
     architecture. Certain semantics of the basic socket
     abstractions are protocol specific. All protocols are
     expected to support the basic model for their particular
     socket type, but may, in addition, provide non-standard
     facilities or extensions to a mechanism. For example, a
     protocol supporting the SOCK_STREAM abstraction may allow
     more than one byte of out-of-band data to be transmitted per
     out-of-band message.

     A network interface is similar to a device interface. Net-
     work interfaces comprise the lowest layer of the networking
     subsystem, interacting with the actual transport hardware.
     An interface may support one or more protocol families
     and/or address formats. The SYNOPSIS section of each

Printed 6/17/012 August 1, 1987   1

network interface entry gives a sample specification of the
related drivers for use in providing a system description to
the /sys/conf/config script. The DIAGNOSTICS section lists
messages which may appear on the console and/or in the sys-
tem error log, /usr/adm/messages (see syslogd(8)), due to
errors in device operation.

PROTOCOLS
The system currently supports the DARPA Internet protocols
and the Xerox Network Systems(tm) protocols. Raw socket
interfaces are provided to the IP protocol layer of the
DARPA Internet, to the IMP link layer (1822), and to the IDP
protocol of Xerox NS. Consult the appropriate manual pages
in this section for more information regarding the support
for each protocol family.

ADDRESSING
Associated with each protocol family is an address format.
The following address formats are used by the system (and
additional formats are defined for possible future implemen-
tation):

```
#define AF_UNIX        1       /* local to host (pipes, portals) */
#define AF_INET        2       /* internetwork: UDP, TCP, etc. */
#define AF_IMPLINK     3       /* arpanet imp addresses */
#define AF_PUP         4       /* pup protocols: e.g. BSP */
#define AF_NS          6       /* Xerox NS protocols */
#define AF_HYLINK      15      /* NSC Hyperchannel */
```

ROUTING
The network facilities provided limited packet routing. A
simple set of data structures comprise a ``routing table''
used in selecting the appropriate network interface when
transmitting packets. This table contains a single entry
for each route to a specific network or host. A user pro-
cess, the routing daemon, maintains this data base with the
aid of two socket-specific ioctl(2) commands, SIOCADDRT and
SIOCDELRT. The commands allow the addition and deletion of
a single routing table entry, respectively. Routing table
manipulations may only be carried out by super-user.

A routing table entry has the following form, as defined in
<net/route.h>;

```
struct rtentry {
    u_long    rt_hash;
    struct    sockaddr rt_dst;
    struct    sockaddr rt_gateway;
    short     rt_flags;
    short     rt_refcnt;
    u_long    rt_use;
    struct    ifnet *rt_ifp;
```

Printed 6/17/012   August 1, 1987   2

```
        };

        with rt_flags defined from,

        #define RTF_UP         0x1    /* route usable */
        #define RTF_GATEWAY    0x2    /* destination is a gateway */
        #define RTF_HOST       0x4    /* host entry (net otherwise) */
        #define RTF_DYNAMIC    0x10   /* created dynamically (by redirect) */
```

Routing table entries come in three flavors: for a specific
host, for all hosts on a specific network, for any destina-
tion not matched by entries of the first two types (a wild-
card route). When the system is booted and addresses are
assigned to the network interfaces, each protocol family
installs a routing table entry for each interface when it is
ready for traffic. Normally the protocol specifies the
route through each interface as a ``direct'' connection to
the destination host or network. If the route is direct,
the transport layer of a protocol family usually requests
the packet be sent to the same host specified in the packet.
Otherwise, the interface is requested to address the packet
to the gateway listed in the routing entry (i.e. the packet
is forwarded).

Routing table entries installed by a user process may not
specify the hash, reference count, use, or interface fields;
these are filled in by the routing routines. If a route is
in use when it is deleted (rt_refcnt is non-zero), the rout-
ing entry will be marked down and removed from the routing
table, but the resources associated with it will not be
reclaimed until all references to it are released. The rout-
ing code returns EEXIST if requested to duplicate an exist-
ing entry, ESRCH if requested to delete a non-existent
entry, or ENOBUFS if insufficient resources were available
to install a new route. User processes read the routing
tables through the /dev/kmem device. The rt_use field con-
tains the number of packets sent along the route.

When routing a packet, the kernel will first attempt to find
a route to the destination host. Failing that, a search is
made for a route to the network of the destination.
Finally, any route to a default (``wildcard'') gateway is
chosen. If multiple routes are present in the table, the
first route found will be used. If no entry is found, the
destination is declared to be unreachable.

A wildcard routing entry is specified with a zero destina-
tion address value. Wildcard routes are used only when the
system fails to find a route to the destination host and
network. The combination of wildcard routes and routing
redirects can provide an economical mechanism for routing
traffic.

INTERFACES
     Each network interface in a system corresponds to a path
     through which messages may be sent and received. A network
     interface usually has a hardware device associated with it,
     though certain interfaces such as the loopback interface,
     lo(4), do not.

     The following ioctl calls may be used to manipulate network
     interfaces. The ioctl is made on a socket (typically of
     type SOCK_DGRAM) in the desired domain. Unless specified
     otherwise, the request takes an ifrequest structure as its
     parameter. This structure has the form

```
struct    ifreq {
#define   IFNAMSIZ 16
char ifr_name[IFNAMSIZ];     /* if name, e.g. "en0" */
union {
    struct  sockaddr ifru_addr;
    struct  sockaddr ifru_dstaddr;
    struct  sockaddr ifru_broadaddr;
    short   ifru_flags;
    int  ifru_metric;
    caddr_t ifru_data;
} ifr_ifru;
#define   ifr_addr  ifr_ifru.ifru_addr          /* address */
#define   ifr_dstaddr    ifr_ifru.ifru_dstaddr  /* other end of p-to-p link */
#define   ifr_broadaddr  ifr_ifru.ifru_broadaddr /* broadcast address */
#define   ifr_flags ifr_ifru.ifru_flags         /* flags */
#define   ifr_metric     ifr_ifru.ifru_metric   /* metric */
#define   ifr_data  ifr_ifru.ifru_data          /* for use by interface */
};
```

     SIOCSIFADDR
        Set interface address for protocol family. Following
        the address assignment, the ``initialization'' routine
        for the interface is called.

     SIOCGIFADDR
        Get interface address for protocol family.

     SIOCSIFDSTADDR
        Set point to point address for protocol family and
        interface.

     SIOCGIFDSTADDR
        Get point to point address for protocol family and
        interface.

     SIOCSIFBRDADDR
        Set broadcast address for protocol family and inter-
        face.

       SIOCGIFBRDADDR
          Get broadcast address for protocol family and inter-
          face.

       SIOCSIFFLAGS
          Set interface flags field. If the interface is marked
          down, any processes currently routing packets through
          the interface are notified; some interfaces may be
          reset so that incoming packets are no longer received.
          When marked up again, the interface is reinitialized.

       SIOCGIFFLAGS
          Get interface flags.

       SIOCSIFMETRIC
          Set interface routing metric. The metric is used only
          by user-level routers.

       SIOCGIFMETRIC
          Get interface metric.

       SIOCGIFCONF
          Get interface configuration list. This request takes
          an ifconf structure (see below) as a value-result
          parameter. The ifc_len field should be initially set
          to the size of the buffer pointed to by ifc_buf. On
          return it will contain the length, in bytes, of the
          configuration list.

       /*
        * Structure used in SIOCGIFCONF request.
        * Used to retrieve interface configuration
        * for machine (useful for programs which
        * must know all networks accessible).
        */
       struct   ifconf {
         int ifc_len;   /* size of associated buffer */
         union {
             caddr_t   ifcu_buf;
             struct   ifreq *ifcu_req;
         } ifc_ifcu;
       #define   ifc_buf ifc_ifcu.ifcu_buf   /* buffer address */
       #define   ifc_req ifc_ifcu.ifcu_req   /* array of structures returned */
       };

SEE ALSO
     socket(2), ioctl(2), intro(4), config(8), routed(8C)

NAME
     ns - Xerox Network Systems(tm) protocol family

SYNOPSIS
     Not currently supported under 2.11BSD

DESCRIPTION
     The NS protocol family is a collection of protocols layered
     atop the Internet Datagram Protocol (IDP) transport layer,
     and using the Xerox NS address formats. The NS family pro-
     vides protocol support for the SOCK_STREAM, SOCK_DGRAM,
     SOCK_SEQPACKET, and SOCK_RAW socket types; the SOCK_RAW
     interface is a debugging tool, allowing you to trace all
     packets entering, (or with toggling kernel variable, addi-
     tionally leaving) the local host.

ADDRESSING
     NS addresses are 12 byte quantities, consisting of a 4 byte
     Network number, a 6 byte Host number and a 2 byte port
     number, all stored in network standard format. (on the VAX
     these are word and byte reversed; on the Sun they are not
     reversed). The include file <netns/ns.h> defines the NS
     address as a structure containing unions (for quicker com-
     parisons).

     Sockets in the Internet protocol family use the following
     addressing structure:

     struct sockaddr_ns {
        short   sns_family;
        struct ns_addr sns_addr;
        char   sns_zero[2];
     };

     where an ns_addr is composed as follows:

     union ns_host {
        u_char   c_host[6];
        u_short   s_host[3];
     };

     union ns_net {
        u_char   c_net[4];
        u_short   s_net[2];
     };

     struct ns_addr {
        union ns_net   x_net;
        union ns_host x_host;
        u_short   x_port;
     };

Printed 6/17/012 January 27, 1996   1

Sockets may be created with an address of all zeroes to
effect ``wildcard'' matching on incoming messages. The
local port address specified in a bind(2) call is restricted
to be greater than NSPORT_RESERVED (=3000, in <netns/ns.h>)
unless the creating process is running as the super-user,
providing a space of protected port numbers.

PROTOCOLS
     The NS protocol family supported by the operating system is
     comprised of the Internet Datagram Protocol (IDP) idp(4P),
     Error Protocol (available through IDP), and Sequenced Packet
     Protocol (SPP) spp(4P).

     SPP is used to support the SOCK_STREAM and SOCK_SEQPACKET
     abstraction, while IDP is used to support the SOCK_DGRAM
     abstraction. The Error protocol is responded to by the ker-
     nel to handle and report errors in protocol processing; it
     is, however, only accessible to user programs through heroic
     actions.

SEE ALSO
     intro(3), byteorder(3N), gethostbyname(3N), getnetent(3N),
     getprotoent(3N), getservent(3N), ns(3N), intro(4N), spp(4P),
     idp(4P), nsip(4)
     Internet Transport Protocols, Xerox Corporation document
     XSIS-028112
     An Advanced 4.3BSD Interprocess Communication Tutorial

NAME
     nsip - software network interface encapsulating ns packets
     in ip packets.

SYNOPSIS
     #include <netns/ns_if.h>

     Not currently supported under 2.11BSD

DESCRIPTION
     The nsip interface is a software mechanism which may be used
     to transmit Xerox NS(tm) packets through otherwise
     uncooperative networks. It functions by prepending an IP
     header, and resubmitting the packet through the unix IP
     machinery.

     The super-user can advise the operating system of a willing
     partner by naming an IP address to be associated with an NS
     address. Presently, only specific hosts pairs are allowed,
     and for each host pair, an artificial point-to-point inter-
     face is constructed. At some future date, IP broadcast
     addresses or hosts may be paired with NS networks or hosts.

     Specifically, a socket option of SO_NSIP_ROUTE is set on a
     socket of family AF_NS, type SOCK_DGRAM, passing the follow-
     ing structure:

     struct nsip_req {
        struct sockaddr rq_ns;   /* must be ns format destination */
        struct sockaddr rq_ip;   /* must be ip format gateway */
        short rq_flags;
     };

DIAGNOSTICS
     nsip%d: can't handle af%d. The interface was handed a mes-
     sage with addresses formatted in an unsuitable address fam-
     ily; the packet was dropped.

SEE ALSO
     intro(4N), ns(4F)

BUGS
     It is absurd to have a separate pseudo-device for each pt-
     to-pt link. There is no way to change the IP address for an
     NS host once the the encapsulation interface is set up. The
     request should honor flags of RTF_GATEWAY to indicate remote
     networks, and the absence of RTF_UP should be a clue to
     remove that partner. This was intended to postpone the
     necessity of rewriting reverse ARP for the en device, and to
     allow passing XNS packets through an Arpanet-Milnet gateway,
     to facilitate testing between some co-operating

rinted 6/17/012   January 27, 1996   1

universities.

NAME
     null - data sink

SYNOPSIS
     major device number(s):
        raw: 1
     minor device encoding:
        mem: 0; kmem: 1; null: 2

DESCRIPTION
     Data written on a null special file is discarded.

     Reads from a null special file always return 0 bytes.

FILES
     /dev/null
     /dev/MAKEDEV script to create special files
     /dev/MAKEDEV.local script to localize special files

SEE ALSO
     mem(4)

NAME
     pty - pseudo terminal driver

SYNOPSIS
     /sys/conf/SYSTEM:
         NPTY ptys # pseudo-terminals, in groups of 8

DESCRIPTION
     The pty driver provides support for a device-pair termed a
     pseudo terminal. A pseudo terminal is a pair of character
     devices, a master device and a slave device. The slave dev-
     ice provides processes an interface identical to that
     described in tty(4). However, whereas all other devices
     which provide the interface described in tty(4) have a
     hardware device of some sort behind them, the slave device
     has, instead, another process manipulating it through the
     master half of the pseudo terminal. That is, anything writ-
     ten on the master device is given to the slave device as
     input and anything written on the slave device is presented
     as input on the master device.

     In configuring, ptys specifies the number of pseudo terminal
     pairs are configured.

     The following ioctl calls apply only to pseudo terminals:

     TIOCSTOP
        Stops output to a terminal (e.g. like typing ^S).
        Takes no parameter.

     TIOCSTART
        Restarts output (stopped by TIOCSTOP or by typing ^S).
        Takes no parameter.

     TIOCPKT
        Enable/disable packet mode. Packet mode is enabled by
        specifying (by reference) a nonzero parameter and dis-
        abled by specifying (by reference) a zero parameter.
        When applied to the master side of a pseudo terminal,
        each subsequent read from the terminal will return data
        written on the slave part of the pseudo terminal pre-
        ceded by a zero byte (symbolically defined as
        TIOCPKT_DATA), or a single byte reflecting control
        status information. In the latter case, the byte is an
        inclusive-or of zero or more of the bits:

        TIOCPKT_FLUSHREAD
              whenever the read queue for the terminal is
              flushed.

        TIOCPKT_FLUSHWRITE
              whenever the write queue for the terminal is

Printed 6/17/012   May 19, 1986   1

flushed.

TIOCPKT_STOP
     whenever output to the terminal is stopped a la
     ^S.

TIOCPKT_START
     whenever output to the terminal is restarted.

TIOCPKT_DOSTOP
     whenever t_stopc is ^S and t_startc is ^Q.

TIOCPKT_NOSTOP
     whenever the start and stop characters are not
     ^S/^Q.

While this mode is in use, the presence of control
status information to be read from the master side may
be detected by a select for exceptional conditions.

This mode is used by rlogin(1C) and rlogind(8C) to
implement a remote-echoed, locally ^S/^Q flow-
controlled remote login with proper back-flushing of
output; it can be used by other similar programs.

TIOCUCNTL
   Enable/disable a mode that allows a small number of
   simple user ioctl commands to be passed through the
   pseudo-terminal, using a protocol similar to that of
   TIOCPKT. The TIOCUCNTL and TIOCPKT modes are mutually
   exclusive. This mode is enabled from the master side
   of a pseudo terminal by specifying (by reference) a
   nonzero parameter and disabled by specifying (by refer-
   ence) a zero parameter. Each subsequent read from the
   master side will return data written on the slave part
   of the pseudo terminal preceded by a zero byte, or a
   single byte reflecting a user control operation on the
   slave side. A user control command consists of a spe-
   cial ioctl operation with no data; the command is given
   as UIOCCMD(n), where n is a number in the range 1-255.
   The operation value n will be received as a single byte
   on the next read from the master side. The ioctl
   UIOCCMD(0) is a no-op that may be used to probe for the
   existence of this facility. As with TIOCPKT mode, com-
   mand operations may be detected with a select for
   exceptional conditions.

TIOCREMOTE
   A mode for the master half of a pseudo terminal,
   independent of TIOCPKT. This mode causes input to the
   pseudo terminal to be flow controlled and not input
   edited (regardless of the terminal mode). Each write

to the control terminal produces a record boundary for
the process reading the terminal. In normal usage, a
write of data is like the data typed as a line on the
terminal; a write of 0 bytes is like typing an end-of-
file character. TIOCREMOTE can be used when doing
remote line editing in a window manager, or whenever
flow controlled input is required.

FILES
    /dev/pty[p-r][0-9a-f]   master pseudo terminals
    /dev/tty[p-r][0-9a-f]   slave pseudo terminals

DIAGNOSTICS
    None.

NAME
     qe - DEC DEQNA Q-bus 10 Mb/s Ethernet interface

SYNOPSIS
     /sys/conf/SYSTEM:
       NQE qe_controllers # DEQNA

DESCRIPTION
     The qe interface provides access to a 10 Mb/s Ethernet net-
     work through the DEC DEQNA Q-bus controller.

     Each of the host's network addresses is specified at boot
     time with an SIOCSIFADDR ioctl. The qe interface employs
     the address resolution protocol described in arp(4P) to map
     dynamically between Internet and Ethernet addresses on the
     local network.

     The interface normally tries to use a ``trailer'' encapsula-
     tion to minimize copying data on input and output. The use
     of trailers is negotiated with ARP. This negotiation may be
     disabled, on a per-interface basis, by setting the
     IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

DIAGNOSTICS
SEE ALSO
     inet(4F), intro(4N), arp(4P)

NAME
     ra - MSCP disk controller interface

SYNOPSIS
     /sys/conf/SYSTEM:
        NRAC ra_controllers # NRAD controllers
        NRAD ra_drives   # RX33/50, RC25, RD51/52/53/54, RA60/80/81/82

     /etc/dtab:
        #Name Unit# Addr   Vector Br Handler(s)   # Comments
        ra ?   172150 154   5   raintr   # uda50, rqdx1/2/3

     major device number(s):
        raw: 14
        block: 5
     minor device encoding:
        bits 0007 specify partition of RA drive
        bits 0070 specify RA drive
        bits 0300 specify RA controller

DESCRIPTION
     This is a driver for the DEC UDA-50 disk controller and for
     other compatible controllers. The UDA-50 communicates with
     the host through a packet oriented protocol termed the Mass
     Storage Control Protocol (MSCP). Consult the file
     <pdp/mscp.h> for a detailed description of this protocol.

     Files with minor device numbers 0 through 7 refer to various
     portions of drive 0; minor devices 8 through 15 refer to
     drive 1, etc. The standard device names begin with ``ra''
     followed by the drive number and then a letter a-h for par-
     titions 0-7 respectively.   The character ? stands here for a
     drive number in the range 0-7.

     The block files access the disk via the system's normal
     buffering mechanism and may be read and written without
     regard to physical disk records. There is also a `raw'
     interface which provides for direct transmission between the
     disk and the user's read or write buffer.   A single read or
     write call results in exactly one I/O operation and there-
     fore raw I/O is considerably more efficient when many words
     are transmitted. The names of the raw files conventionally
     begin with an extra `r.'

     In raw I/O the buffer must begin on a word (even) boundary,
     and counts should be a multiple of 512 bytes (a disk sec-
     tor). Likewise seek calls should specify a multiple of 512
     bytes.

DISK SUPPORT
     This driver configures the drive type of each drive when it
     is first opened. Partition information is read from the

disklabel. If there is no label or the label is corrupt
then the 'a' partition is used to span the entire drive.

The ra?a partition is normally used for the root file sys-
tem, the ra?b partition as a swap area, and the ra?c parti-
tion for pack-pack copying (it maps the entire disk).

FILES
    /dev/ra[0-7][a-h]
    /dev/rra[0-7][a-h]
    /dev/MAKEDEV script to create special files

SEE ALSO
    hk(4), ram(4), rk(4), rl(4), rp(4), rx(4), si(4), xp(4),
    dtab(5), autoconfig(8), disklabel(8)

DIAGNOSTICS
    rasa %o, state %d. (Additional status information given
    after a hard i/o error.) The values of the UDA-50 status
    register and the internal driver state are printed.

    ra%d: interrupt in unknown state %d ignored. An interrupt
    was received when the driver was in an unknown internal
    state. Indicates a hardware problem or a driver bug.

    ra%d: fatal error (%o). The UDA-50 indicated a ``fatal
    error'' in the status returned to the host. The contents of
    the status register are displayed.

    ra%d,%d: OFFLINE. (Additional status information given
    after a hard i/o error.) A hard i/o error occurred because
    the drive was not on-line. The attached unit number and the
    MSCP unit numbers are printed.

    status %o. (Additional status information given after a
    hard i/o error.) The status information returned from the
    UDA-50 is tacked onto the end of the hard error message
    printed on the console.

    ra: unknown packet opcode=0%o. An MSCP packet of unknown
    type was received from the UDA-50. Check the cabling to the
    controller.

    The following errors are interpretations of MSCP error mes-
    sages returned by the UDA-50 to the host.

    ra: %s error, controller error, event 0%o.

    ra: %s error, host memory access error, event 0%o, addr 0%o.

    ra: %s error, disk transfer error, unit %d, grp 0x%x, hdr
    0x%x.

Printed 6/17/012 September 6, 1987   2

ra: %s error, SDI error, unit %d, event 0%o, hdr 0x%x.

ra: %s error, small disk error, unit %d, event 0%o, cyl %d.

ra: %s error, unknown error, unit %d, format 0%o, event 0%o.

BUGS

In raw I/O read and write(2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek(2) should always deal in 512-byte multiples.

NAME
     ram - ram disk driver

SYNOPSIS
     /sys/conf/SYSTEM:
        NRAM ram_size # RAM disk size (512-byte blocks)

     major device number(s):
        block: 3
     minor device encoding:
        must be zero (0)

DESCRIPTION
     The ram pseudo-device provides a very fast extended memory
     store. It's use is intended for file systems like /tmp and
     applications which need to access a reasonably large amount
     of data quickly.

     The amount of memory dedicated to the ram device is con-
     trolled by the NRAM definition in units of 512-byte blocks.
     This is also patchable in the system binary through the
     variable ram_size (though a patched system would have to be
     rebooted before any change took effect; see adb(1)). This
     makes it easy to test the effects of different ram disk
     sizes on system performance. It's important to note that
     any space given to the ram device is permanently allocated
     at system boot time. Dedicating too much memory can
     adversely affect system performance by forcing the system to
     swap heavily as in a memory poor environment.

     The block file accesses the ram disk via the system's
     buffering mechanism through a buffer sharing arrangement
     with the buffer cache. It may be read and written without
     regard to physical disk records. There is no `raw' inter-
     face since no speed advantage is gained by such an interface
     with the ram disk.

DISK SUPPORT
     The ram driver does not support pseudo-disks (partitions).
     The special files refer to the entire `drive' as a single
     sequentially addressed file.

     A typical use for the ram disk would be to mount /tmp on it.
     Note that if this arrangement is recorded in /etc/fstab then
     /etc/rc will have to be modified slightly to do a mkfs(8) on
     the ram disk before the standard file system checks are
     done.

FILES
     /dev/ram   block file
     /dev/MAKEDEV script to create special files
     /dev/MAKEDEV.local   script to localize special files

Printed 6/17/012 Januray 27, 1996   1

SEE ALSO
    hk(4), ra(4), rl(4), rk(4), rp(4), rx(4), si(4), xp(4)
    dtab(5), autoconfig(8)

DIAGNOSTICS
    ram: no space. There is not enough memory to allocate the
    space needed by the ram disk. The ram disk is disabled.
    Any attempts to access it will return an error.

    ram: not allocated. No memory was allocated to the ram disk
    and an attempt was made to open it. Either not enough
    memory was available at boot time or the kernel variable
    ram_size was set to zero.

BUGS
    The ram driver is only available under 2.11BSD.

NAME
     rk - RK-11/RK05 disk

SYNOPSIS
     /sys/conf/SYSTEM:
        NRK   rk_drives # RK05

     /etc/dtab:
        #Name   Unit# Addr   Vector Br Handler(s)   # Comments
        rk ?   177400 220   5 rkintr   # rk05

     major device number(s):
        raw: 15
        block: 6
     minor device encoding:
        specifies drive: <rk_drive>

DESCRIPTION
     Minor device numbers are drive numbers on one controller.
     The standard device names begin with ``rk'' followed by the
     drive number and then the letter "h". The character ?
     stands here for a drive number in the range 0-7.

     The block files access the disk via the system's normal
     buffering mechanism and may be read and written without
     regard to physical disk records. There is also a `raw'
     interface which provides for direct transmission between the
     disk and the user's read or write buffer.   A single read or
     write call results in exactly one I/O operation and there-
     fore raw I/O is considerably more efficient when many words
     are transmitted. The names of the raw files conventionally
     begin with an extra `r.'

     In raw I/O the buffer must begin on a word (even) boundary,
     and counts should be a multiple of 512 bytes (a disk sec-
     tor). Likewise seek calls should specify a multiple of 512
     bytes.

DISK SUPPORT
     The rk driver does not support pseudo-disks (partitions).
     Each file rk?h refers to the entire drive as a single
     sequentially addressed file. Each drive has 4872 512-byte
     blocks.

     It's not clear what one would do with one of these drives if
     one had one ...

FILES
     /dev/rk[0-7]h   block files
     /dev/rrk[0-7]h   raw files
     /dev/MAKEDEV   script to create special files
     /dev/MAKEDEV.local   script to localize special files

Printed 6/17/012 August 20, 1987   1

SEE ALSO
    hk(4), ra(4), ram(4), rl(4), rp(4), rx(4), si(4), xp(4),
    dtab(5), autoconfig(8)

DIAGNOSTICS
    rk%d: hard error sn%d er=%b ds=%b. An unrecoverable error
    occurred during transfer of the specified sector of the
    specified disk. The contents of the two error registers are
    also printed in octal and symbolically with bits decoded.
    The error was either unrecoverable, or a large number of
    retry attempts could not recover the error.

    rk%d: write locked. The write protect switch was set on the
    drive when a write was attempted. The write operation is
    not recoverable.

BUGS
    In raw I/O read and write(2) truncate file offsets to 512-
    byte block boundaries, and write scribbles on the tail of
    incomplete blocks. Thus, in programs that are likely to
    access raw devices, read, write and lseek(2) should always
    deal in 512-byte multiples.

    DEC-standard error logging should be supported.

    A program to analyze the logged error information (even in
    its present reduced form) is needed.

NAME
     rl - RL-11/RL01, RL02 moving-head disk

SYNOPSIS
     /sys/conf/SYSTEM:
        NRL   rl_drives # RL01/02

     /etc/dtab:
        #Name   Unit# Addr   Vector Br Handler(s)   # Comments
        rl ?   174400 160   5   rlintr   # rl-01/02

     major device number(s):
        raw: 16
        block: 7
     minor device encoding:
        bits 0007 specify partition of RL drive
        bits 0070 specify RL drive

DESCRIPTION
     The block files access the disk via the system's normal
     buffering mechanism and may be read and written without
     regard to physical disk records. There is also a `raw'
     interface which provides for direct transmission between the
     disk and the user's read or write buffer.   A single read or
     write call results in exactly one I/O operation and there-
     fore raw I/O is considerably more efficient when many words
     are transmitted. The names of the raw files conventionally
     begin with an extra `r.'

     In raw I/O the buffer must begin on a word (even) boundary,
     and counts should be a multiple of 512 bytes (a disk sec-
     tor). Likewise seek calls should specify a multiple of 512
     bytes.

DISK SUPPORT
     The RL01 drives are each 10240 blocks long and the RL02
     drives are 20480 blocks long.

     On a RL02 there is room for a full sized root ('a') parti-
     tion and a reasonable sized swap ('b') partition.   The RL01
     can only (realistically) have a single 5mb partition.

FILES
     /dev/rl[0-3][a-h] block files
     /dev/rrl[0-3][a-h] raw files
     /dev/MAKEDEV script to create special files
     /dev/MAKEDEV.local script to localize special files

SEE ALSO
     hk(4), ra(4), ram(4), rk(4), rp(4), rx(4), si(4), xp(4),
     dtab(5), autoconfig(8)

DIAGNOSTICS
     rl%d: hard error sn%d cs=%b da=%b. An unrecoverable error
     occurred during transfer of the specified sector of the
     specified disk. The contents of the two error registers are
     also printed in octal and symbolically with bits decoded.
     The error was either unrecoverable, or a large number of
     retry attempts could not recover the error.

     rl%d: hard error sn%d mp=%b da=%b. An unrecoverable drive
     error occured during transfer of the specified sector of the
     specified disk. The contents of the two error registers are
     also printed in octal and symbolically with bits decoded.
     The error was either unrecoverable, or a large number of
     retry attempts could not recover the error.

     rl%d: write locked. The write protect switch was set on the
     drive when a write was attempted. The write operation is
     not recoverable.

     rl%d: can't get status. A ``get status'' command on the
     specified drive failed. The error is unrecoverable.

BUGS
     In raw I/O read and write(2) truncate file offsets to 512-
     byte block boundaries, and write scribbles on the tail of
     incomplete blocks. Thus, in programs that are likely to
     access raw devices, read, write and lseek(2) should always
     deal in 512-byte multiples.

     DEC-standard error logging should be supported.

     A program to analyze the logged error information (even in
     its present reduced form) is needed.

NAME
     rx, rb, rz - XMODEM, YMODEM, ZMODEM (Batch) file receive

SYNOPSIS
     rz [- +abepqtuvy]
     rb [- +abqtuvy]
     rx [- abceqtuv] file
     gz file ...
     [-][v]rzCOMMAND

DESCRIPTION
     This program uses error correcting protocols to receive
     files over a dial-in serial port from a variety of programs
     running under PC-DOS, CP/M, Unix, and other operating sys-
     tems. It is invoked from a shell prompt manually, or
     automatically as a result of an "sz file ..." command given
     to the calling program.

     Rz is not intended be called from cu(1), or other communica-
     tions programs. Unix flavors of Omen Technology's
     Professional-YAM communications software are available for
     dial-out applications. (Reg.)

     Rz (Receive ZMODEM) receives files with the ZMODEM batch
     protocol.   Pathnames are supplied by the sending program,
     and directories are made if necessary (and possible). Nor-
     mally, the "rz" command is automatically issued by the cal-
     ling ZMODEM program, but some defective ZMODEM implementa-
     tions may require starting rz the old fashioned way. Rz
     does not support ZMODEM Crash Recovery and certain other
     ZMODEM features. Unix flavors of Professional-YAM may be
     linked to "rz" and used in place of this program to support
     these ZMODEM features.

     Rb receives file(s) with YMODEM, accepting either standard
     128 byte sectors or 1024 byte sectors (YAM sb -k option).
     The user should determine when the 1024 byte block length
     actually improves throughput without causing lost data or
     even system crashes.

     If True YMODEM (Omen Technology trademark) file information
     (file length, etc.) is received, the file length controls
     the number of bytes written to the output dataset, and the
     modify time and file mode (iff non zero) are set accord-
     ingly.

     If no True YMODEM file information is received, slashes in
     the pathname are changed to underscore, and any trailing
     period in the pathname is eliminated. This conversion is
     useful for files received from CP/M systems. With YMODEM,

Printed 6/17/012   OMEN   1

each file name is converted to lower case unless it contains
one or more lower case letters.

Rx receives a single file with XMODEM or XMODEM-1k protocol.
The user should determine when the 1024 byte block length
actually improves throughput without causing problems. The
user must supply the file name to both sending and receiving
programs.   Up to 1023 garbage characters may be added to the
received file.

Gz is a shell script which calls sz to command Pro-YAM or
ZCOMM to transmit the specified files. Pathnames used with
gz must be escaped if they have special significance to the
Unix shell.
EXAMPLE: gz "-a C:*.c D:*.h"

Rz may be invoked as rzCOMMAND (with an optional leading -
as generated by login(1)). For each received file, rz will
pipe the file to ``COMMAND filename'' where filename is the
name of the transmitted file with the file contents as stan-
dard input.

Each file transfer is acknowledged when COMMAND exits with 0
status. A non zero exit status terminates transfers.

A typical use for this form is rzrmail which calls rmail(1)
to post mail to the user specified by the transmitted file
name. For example, sending the file "caf" from a PC-DOS
system to rzrmail on a Unix system would result in the con-
tents of the DOS file "caf" being mailed to user "caf".

On some Unix systems, the login directory must contain a
link to COMMAND as login sets SHELL=rsh which disallows
absolute pathnames. If invoked with a leading ``v'', rz
will report progress to /tmp/rzlog. The following entry
works for Unix SYS III/V:
        rzrmail::5:1::/bin:/usr/local/rzrmail
If the SHELL environment variable includes rsh or rksh (res-
tricted shell), rz will not accept absolute pathnames or
references to a parent directory, will not modify an exist-
ing file, and removes any files received in error.

If rz is invoked with stdout and stderr to different
datasets, Verbose is set to 2, causing frame by frame pro-
gress reports to stderr. This may be disabled with the q
option.

The meanings of the available options are:

    a   Convert files to Unix conventions by stripping carriage
       returns and all characters beginning with the first
       Control Z (CP/M end of file).
    b   Binary (tell it like it is) file transfer override.
    c   Request 16 bit CRC. XMODEM file transfers default to 8
       bit checksum. YMODEM and ZMODEM normally use 16 bit
       CRC.
    D   Output file data to /dev/null; for testing. (Unix
       only)
    e   Force sender to escape all control characters; normally
       XON, XOFF, DLE, CR-@-CR, and Ctrl-X are escaped.
    p   (ZMODEM) Protect: skip file if destination file exists.
    q   Quiet suppresses verbosity.
    t tim
       Change timeout to tim tenths of seconds.
    v   Verbose causes a list of file names to be appended to
       /tmp/rzlog .   More v's generate more output.
    y   Yes, clobber any existing files with the same name.

EXAMPLES
    (Pro-YAM command)
    <ALT-2>
    Pro-YAM Command: sz *.h *.c
    (This automatically invokes rz on the connected system.)

SEE ALSO
    ZMODEM.DOC, YMODEM.DOC, Professional-YAM, crc(omen),
    sz(omen), usq(omen), undos(omen)

    Compile time options required for various operating systems
    are described in the source file.

NOTES
    ZMODEM's support of XOFF/XON flow control allows proper
    operation in many environments that do not support XMODEM
    uploads. Unfortunately, not all timesharing systems support
    input flow control. The TTY input buffering on some systems
    may not adequately buffer long blocks or streaming input at
    high speed. You should suspect this problem when you can't
    send data to the Unix system at high speeds using ZMODEM,
    but YMODEM-1k or XMODEM-1k, when YMODEM with 128 byte blocks
    works properly.

    The DSZ or Pro-YAM zmodem l numeric parameter may be set to
    a value between 64 and 1024 to limit the burst length ("zmo-
    dem pl128"). Although this compromises ZMODEM's throughput,
    ZMODEM's superior reliability remains intact.

    If a program that does not properly implement the specified
    file transfer protocol causes rz to "hang" the port after a
    failed transfer, either wait for rz to time out or keyboard
    a dozen Ctrl-X characters. Every reported instance of this

problem has been corrected by using ZCOMM, Pro-YAM, DSZ, or other program with a correct implementation of the specified protocol.

Many programs claiming to support YMODEM only support XMODEM with 1k blocks, and they often don't get that quite right.

In the case of a few poorly designed microcomputers, sending serial data to a tty port at sustained high speeds has been known to cause lockups, system halts, kernel panics, and occasional antisocial behaviour. This problem is not unique to rz; CRT terminals with block mode transmission and line noise have the same effect. When experimenting with high speed input to a system, consider rebooting the system if the file transfers are not successful, especially if the personality of the system appears altered.

The Unix "ulimit" parameter must be set high enough to permit large file transfers to Unix.

32 bit CRC code courtesy Gary S. Brown. Directory creation code from John Gilmore's PD TAR program.

BUGS
     Rz is not intended be called from cu(1), or other communications programs. Unix flavors of Omen Technology's Professional-YAM communications software are available for dial-out applications.

     Rz does not support ZMODEM Crash Recovery and many other ZMODEM features. Unix flavors of Professional-YAM may be linked to "rz" to support these features.

     Pathnames are restricted to 127 characters. In XMODEM single file mode, the pathname given on the command line is still processed as described above. The ASCII option's CR/LF to NL translation merely deletes CR's; undos(omen) performs a more intelligent translation.

VMS VERSION
     The VMS version does not set the file time.

     VMS C Standard I/O and RMS may interact to modify file contents unexpectedly.

     The VMS version does not support invocation as rzCOMMAND .
     The current VMS version does not support XMODEM, XMODEM-1k, or YMODEM.

     According to the VMS documentation, the buffered input routine used on the VMS version of rz introduces a delay of up to one second for each protocol transaction. This delay may

be significant for very short files. Removing the "#define
BUFREAD" line from rz.c will eliminate this delay at the
expense of increased CPU utilization.

For high speed operation, try increasing the SYSGEN parame-
ter TTY_TYPAHDSZ to 256.

The VMS version causes DCL to generate a random off the wall
error message under some error conditions; this is a result
of the incompatibility of the VMS "exit" function with the
Unix/MSDOS standard.

ZMODEM CAPABILITIES
     Rz supports incoming ZMODEM binary (-b), ASCII (-a), protect
     (-p), clobber (-y), and append (-+) requests. Other options
     sent by the sender are ignored. The default is protect (-p)
     and binary (-b).

     The Unix versions support ZMODEM command execution.

FILES
     rz.c, crctab.c, rbsb.c, zm.c, zmodem.h Unix source files.

     rz.c, crctab.c, vrzsz.c, zm.c, zmodem.h, vmodem.h,
     vvmodem.c, VMS source files.

     /tmp/rzlog stores debugging output generated with -vv option
     (rzlog on VMS).

NAME
     rx - DEC RX02 floppy disk

SYNOPSIS
     /sys/conf/SYSTEM:
       NRX   rx_units # RX02

     /etc/dtab:
       #Name   Unit# Addr   Vector Br Handler(s)   # Comments
       rx ?   177170 264   5 rxintr   # rx02

     major device number(s):
       raw: 17
       block: 8
     minor device encoding:
       bit 01 specifies drive: <rx_drive>
       bit 02 specifies density: single: 0, double: 1

DESCRIPTION
     The rx device provides access to a DEC RX02 floppy disk unit
     with M8256 interface module (RX211 configuration). The RX02
     uses 8-inch, single-sided, soft-sectored floppy disks (with
     pre-formatted industry-standard headers) in either single or
     double density.

     Floppy disks handled by the RX02 contain 77 tracks, each
     with 26 sectors (for a total of 2,002 sectors). The sector
     size is 128 bytes for single density, 256 bytes for double
     density. Single density disks are compatible with the RX01
     floppy disk unit and with IBM 3740 Series Diskette 1 sys-
     tems.

     Files with minor device numbers 0 and 1 refer to drives 0
     and 1 in single density mode; minor devices 2 and 3 refer to
     drives 0 and 1 in double density mode. The standard device
     names begin with ``rx'' followed by the drive number and
     then a letter a-b for single and double density access to
     the drive respectively. The character ? stands here for a
     drive number in the range 0-1.

     The block files access the disk via the system's normal
     buffering mechanism and may be read and written without
     regard to physical disk records. There is also a `raw'
     interface which provides for direct transmission between the
     disk and the user's read or write buffer.   A single read or
     write call results in exactly one I/O operation and there-
     fore raw I/O is considerably more efficient when many words
     are transmitted. The names of the raw files conventionally
     begin with an extra `r.'

     In raw I/O the buffer must begin on a word (even) boundary,
     and counts should be a multiple of the disk's sector size

Printed 6/17/012 January 27, 1996   1

(either 128 or 256 bytes depending on selected density.)
Likewise seek calls should specify a multiple of the disk's
sector size.

In addition to normal (`block' and `raw') i/o, the driver
supports formatting of disks for either density.

DISK SUPPORT
    The rx driver does not support pseudo-disks (partitions).
    Each file rx?[ab] refers to the entire drive as a single
    sequentially addressed file. The physical disk sector size
    is 128 bytes for single density and 256 bytes for double
    density. The logical block size is 1024 bytes. Each
    diskette has 250 logical blocks in single density and 500
    logical blocks in double density.

    The size and density of the disks are specified as follows:

        disk   minor device unit   density
        rx0a   0   0   single
        rx1a   1   1   single
        rx0b   2   0   double
        rx1b   3   1   double

    Even though the storage capacity on a floppy disk is quite
    small, it is possible to make file systems on them. For
    example, the commands
        % rxformat /dev/rrx0a
        % newfs /dev/rrx0a
    and
        % rxformat /dev/rrx1b
        % newfs /dev/rrx1b
    format and make file systems on the single density disk in
    the left drive and the double density disk in the right
    drive with 241 and 486 kbytes available, respectively, for
    file storage. Using tar(1) gives somewhat more efficient
    utilization of the available space for file storage. The
    RX02 disks are primarily useful for small archives and
    transfer of small systems or distributions.

    An ioctl(2) call is available is available to format RX02
    disks:
        #include <pdpuba/rxreg.h>
        ioctl(fildes, RXIOC_FORMAT)
    The density is determined by the device opened.

FILES
    /dev/rx[01][ab]   block files
    /dev/rrx[01][ab]   raw files
    /dev/MAKEDEV   script to create special files
    /dev/MAKEDEV.local   script to localize special files

Printed 6/17/012 January 27, 1996   2

SEE ALSO
    hk(4), ra(4), ram(4), rk(4), rl(4), rp(4), si(4), xp(4),
    dtab(5), autoconfig(8), rxformat(8V)

DIAGNOSTICS
    rx2%d: hard error sn%d cs=%b er=%b. An unrecoverable error
    occurred during transfer of the specified sector of the
    specified disk. The contents of the two error registers are
    also printed in octal and symbolically with bits decoded.
    The error was either unrecoverable, or a large number of
    retry attempts could not recover the error.

BUGS
    In raw I/O read and write(2) truncate file offsets to disk
    sector size block boundaries (either 128 or 256 bytes
    depending on the selected density), and write scribbles on
    the tail of incomplete blocks. Thus, in programs that are
    likely to access raw devices, read, write and lseek(2)
    should always deal in disk sector size multiples.

    DEC-standard error logging should be supported.

    A program to analyze the logged error information (even in
    its present reduced form) is needed.

    The 4.3BSD rx driver which supports more capabilities should
    be ported to 2.11BSD.

NAME
     si - SI 9500/CDC 9766 moving head disk

SYNOPSIS
     /sys/conf/SYSTEM:
        NSI   si_drives # SI 9500 driver for CDC 9766 disks

     /etc/dtab:
        #Name   Unit# Addr   Vector Br Handler(s)   # Comments
        si ?   176700 170   5 siintr   # si9500

     major device number(s):
        raw: 18
        block: 9
     minor device encoding:
        bits 0007 specify partition of SI drive
        bits 0070 specify SI drive

DESCRIPTION
     Files with minor device numbers 0 through 7 refer to various
     portions of drive 0; minor devices 8 through 15 refer to
     drive 1, etc. The standard device names begin with ``si''
     followed by the drive number and then a letter a-h for par-
     titions 0-7 respectively.   The character ? stands here for a
     drive number in the range 0-7.

     The block files access the disk via the system's normal
     buffering mechanism and may be read and written without
     regard to physical disk records. There is also a `raw'
     interface which provides for direct transmission between the
     disk and the user's read or write buffer.   A single read or
     write call results in exactly one I/O operation and there-
     fore raw I/O is considerably more efficient when many words
     are transmitted. The names of the raw files conventionally
     begin with an extra `r.'

     In raw I/O the buffer must begin on a word (even) boundary,
     and counts should be a multiple of 512 bytes (a disk sec-
     tor). Likewise seek calls should specify a multiple of 512
     bytes.

DISK SUPPORT
     The origin and size (in sectors) of the pseudo-disks on each
     drive are as follows:

SI 9500/CDC9766 partitions:

disk	start	length	cyls	comments
xp?a	0	9120	0 - 14	/
xp?b	9120	9120	15 - 29	swap
xp?c	18240	234080	30 - 414	
xp?d	252320	247906	415 - 822*	
xp?e	18240	164160	30 - 299	/usr
xp?f	182400	152000	300 - 549	
xp?g	334400	165826	550 - 822*	
xp?h	0	500384	0 - 822	whole pack

Those partitions marked with an asterisk (``*'') actually
stop short of the indicated ending cylinder to protect any
bad block forwarding information on the packs. The indi-
cated lengths are correct. Partition ``h'' must be used to
access the bad block forwarding area. N.B.: the si driver
does not support bad block forwarding; the space is reserved
in the event bad block forwarding is ever added to the
driver.

FILES
    /dev/si[0-7][a-h]   block files
    /dev/rsi[0-7][a-h]   raw files
    /dev/MAKEDEV   script to create special files
    /dev/MAKEDEV.local   script to localize special files

SEE ALSO
    hk(4), ra(4), ram(4), rk(4), rl(4), rp(4), rx(4), xp(4),
    dtab(5), autoconfig(8)

DIAGNOSTICS
    si%d%c: hard error sn%d cnr=%b err=%b. An unrecoverable
    error occurred during transfer of the specified sector of
    the specified disk partition. The contents of the two error
    registers are also printed in octal and symbolically with
    bits decoded. The error was either unrecoverable, or a
    large number of retry attempts (including offset positioning
    and drive recalibration) could not recover the error.

    si%d%c: hard error sn%d ssr=%b err=%b. An unrecoverable
    error occurred during transfer of the specified sector of
    the specified disk partition. The contents of the two error
    registers are also printed in octal and symbolically with
    bits decoded. The error was either unrecoverable, or a
    large number of retry attempts (including offset positioning
    and drive recalibration) could not recover the error.

BUGS
    In raw I/O read and write(2) truncate file offsets to 512-
    byte block boundaries, and write scribbles on the tail of
    incomplete blocks. Thus, in programs that are likely to
    access raw devices, read, write and lseek(2) should always

Printed 6/17/012 August 20, 1987   2

deal in 512-byte multiples.

The partition tables for the file systems should be read off
of each pack, as they are never quite what any single
installation would prefer, and this would make packs more
portable.

NAME
     spp - Xerox Sequenced Packet Protocol

SYNOPSIS
     #include <sys/socket.h>
     #include <netns/ns.h>
     s = socket(AF_NS, SOCK_STREAM, 0);

     #include <netns/sp.h>
     s = socket(AF_NS, SOCK_SEQPACKET, 0);

DESCRIPTION
     The SPP protocol provides reliable, flow-controlled, two-way
     transmission of data. It is a byte-stream protocol used to
     support the SOCK_STREAM abstraction. SPP uses the standard
     NS(tm) address formats.

     Sockets utilizing the SPP protocol are either "active" or
     "passive". Active sockets initiate connections to passive
     sockets. By default SPP sockets are created active; to
     create a passive socket the listen(2) system call must be
     used after binding the socket with the bind(2) system call.
     Only passive sockets may use the accept(2) call to accept
     incoming connections. Only active sockets may use the con-
     nect(2) call to initiate connections.

     Passive sockets may "underspecify" their location to match
     incoming connection requests from multiple networks. This
     technique, termed "wildcard addressing", allows a single
     server to provide service to clients on multiple networks.
     To create a socket which listens on all networks, the NS
     address of all zeroes must be bound. The SPP port may still
     be specified at this time; if the port is not specified the
     system will assign one. Once a connection has been esta-
     blished the socket's address is fixed by the peer entity's
     location.   The address assigned the socket is the address
     associated with the network interface through which packets
     are being transmitted and received. Normally this address
     corresponds to the peer entity's network.

     If the SOCK_SEQPACKET socket type is specified, each packet
     received has the actual 12 byte sequenced packet header left
     for the user to inspect:
     struct sphdr {
        u_char   sp_cc;   /* connection control */
     #define   SP_EM   0x10   /* end of message */
        u_char   sp_dt;   /* datastream type */
        u_short   sp_sid;
        u_short   sp_did;
        u_short   sp_seq;
        u_short   sp_ack;
        u_short   sp_alo;

```
      };
      This facilitates the implementation of higher level Xerox
      protocols which make use of the data stream type field and
      the end of message bit.  Conversely, the user is required to
      supply a 12 byte header, the only part of which inspected is
      the data stream type and end of message fields.
```

For either socket type, packets received with the Attention
bit sent are interpreted as out of band data. Data sent
with send(..., ..., ..., MSG_OOB) cause the attention bit to
be set.

DIAGNOSTICS
      A socket operation may fail with one of the following errors
      returned:

      [EISCONN]   when trying to establish a connection on
                            a socket which already has one;

      [ENOBUFS]   when the system runs out of memory for
                            an internal data structure;

      [ETIMEDOUT]   when a connection was dropped due to
                            excessive retransmissions;

      [ECONNRESET] when the remote peer forces the connec-
                            tion to be closed;

      [ECONNREFUSED]   when the remote peer actively refuses
                            connection establishment (usually
                            because no process is listening to the
                            port);

      [EADDRINUSE] when an attempt is made to create a
                            socket with a port which has already
                            been allocated;

      [EADDRNOTAVAIL]   when an attempt is made to create a
                            socket with a network address for which
                            no network interface exists.

SOCKET OPTIONS
      SO_DEFAULT_HEADERS when set, this determines the data
                            stream type and whether the end of mes-
                            sage bit is to be set on every ensuing
                            packet.

      SO_MTU   This specifies the maximum ammount of
                            user data in a single packet.   The
                            default is 576 bytes - sizeof(struct
                            spidp). This quantity affects windowing
                            -- increasing it without increasing the

amount of buffering in the socket will
lower the number of unread packets
accepted. Anything larger than the
default will not be forwarded by a bona
fide XEROX product internetwork router.
The data argument for the setsockopt
call must be an unsigned short.

SEE ALSO
    intro(4N), ns(4F)

BUGS
    There should be some way to reflect record boundaries in a
    stream. For stream mode, there should be an option to get
    the data stream type of the record the user process is about
    to receive.

NAME
     sri - DR11-C IMP interface

SYNOPSIS
     /sys/conf/SYSTEM:
         NSRI sri_controllers   # SRI DR11c ARPAnet IMP

DESCRIPTION
DIAGNOSTICS
     sri%d: not alive. The initialization routine was entered
     even though the device did not autoconfigure. This indi-
     cates a system problem.

     sri%d: can't initialize. Insufficient UNIBUS resources
     existed to initialize the device. This is likely to occur
     when the device is run on a buffered data path on an 11/750
     and other network interfaces are also configured to use buf-
     fered data paths, or when it is configured to use buffered
     data paths on an 11/730 (which has none).

     sri%d: imp doesn't respond, ibf=%b. The driver attempted to
     initialize the device, but the IMP failed to respond after 5
     tries. Check the cabling.

     sri%d: stray xmit interrupt, csr=%b. An interrupt occurred
     when no output had previously been started.

     sri%d: output error, csr=%b. The device indicated a problem
     sending data on output.

     sri%d: input error, ibf=%b. The device indicated a problem
     receiving data on input.

     sri%d: bad length=%d. An input operation resulted in a data
     transfer of less than 10 or more than 1008 bytes of data
     into memory (according to the word count register). This
     should never happen as the maximum size of a host-IMP mes-
     sage is 1018 bytes.

NAME
     swap - swap device

DESCRIPTION
     This file refers to the swap device in use by the system.
     It should be linked to the special file for the disk or disk
     partition on which the system swaps. It is used by programs
     that wish to read from the swap device without knowing its
     real name. The script /dev/MAKEDEV.local normally is edited
     by the system builder to set /dev/swap up correctly.

FILES
     /dev/swap
     /dev/MAKEDEV script to create special files
     /dev/MAKEDEV.local script to localize special files

SEE ALSO
     ps(1), w(1)

NAME
     tb - line discipline for digitizing devices

SYNOPSIS
     /sys/conf/SYSTEM:
        NTB   1   # RS232 interface for Genisco/Hitachi tablets

DESCRIPTION
     This line discipline provides a polled interface to many
     common digitizing devices which are connected to a host
     through a serial line. When these devices stream data at
     high speed, the use of the line discipline is critical in
     minimizing the number of samples that would otherwise be
     lost due to buffer exhaustion in the tty(4) handler.

     The line discipline is enabled by a sequence:

         #include <sys/tablet.h>
         int ldisc = TBLDISC, fildes; ...
         ioctl(fildes, TIOCSETD, &ldisc);

     A typical application program then polls the digitizing dev-
     ice by reading a binary data structure which contains: the
     current X and Y positions (in the device coordinate space),
     up-down status of the buttons or pen stylus, proximity
     information (when available), and a count of the number of
     samples received from the input device since it was opened.
     In addition, devices such as the GTCO append tilt and pres-
     sure information to the end of the aforementioned structure.
     For the Polhemus 3-D digitizer the structure read is com-
     pletely different. Refer to the include file for a complete
     description.

     While in tablet mode, normal teletype input and output func-
     tions take place. Thus, if an 8 bit output data path is
     desired, it is necessary to prepare the output line by put-
     ting it into RAW mode using ioctl(2). This must be done
     before changing the discipline with TIOCSETD, as most
     ioctl(2) calls are disabled while in tablet line-discipline
     mode.

     The line discipline supports ioctl(2) requests to get/set
     the operating mode, and to get/set the tablet type and
     operating mode by or-ing the two values together.

     The line discipline supports digitizing devices which are
     compatible with Hitachi, GTCO, or Polhemus protocol formats.
     For Hitachi there are several formats with that used in the
     newer model HDG-1111B the most common.

SEE ALSO
     tty(4)

DIAGNOSTICS
     None.

NAME
     tcp - Internet Transmission Control Protocol

SYNOPSIS
     #include <sys/socket.h>
     #include <netinet/in.h>

     s = socket(AF_INET, SOCK_STREAM, 0);

DESCRIPTION
     The TCP protocol provides reliable, flow-controlled, two-way
     transmission of data. It is a byte-stream protocol used to
     support the SOCK_STREAM abstraction. TCP uses the standard
     Internet address format and, in addition, provides a per-
     host collection of "port addresses". Thus, each address is
     composed of an Internet address specifying the host and net-
     work, with a specific TCP port on the host identifying the
     peer entity.

     Sockets utilizing the tcp protocol are either "active" or
     "passive". Active sockets initiate connections to passive
     sockets. By default TCP sockets are created active; to
     create a passive socket the listen(2) system call must be
     used after binding the socket with the bind(2) system call.
     Only passive sockets may use the accept(2) call to accept
     incoming connections. Only active sockets may use the con-
     nect(2) call to initiate connections.

     Passive sockets may "underspecify" their location to match
     incoming connection requests from multiple networks. This
     technique, termed "wildcard addressing", allows a single
     server to provide service to clients on multiple networks.
     To create a socket which listens on all networks, the Inter-
     net address INADDR_ANY must be bound. The TCP port may
     still be specified at this time; if the port is not speci-
     fied the system will assign one. Once a connection has been
     established the socket's address is fixed by the peer
     entity's location.   The address assigned the socket is the
     address associated with the network interface through which
     packets are being transmitted and received. Normally this
     address corresponds to the peer entity's network.

     TCP supports one socket option which is set with set-
     sockopt(2) and tested with getsockopt(2).   Under most cir-
     cumstances, TCP sends data when it is presented; when out-
     standing data has not yet been acknowledged, it gathers
     small amounts of output to be sent in a single packet once
     an acknowledgement is received. For a small number of
     clients, such as window systems that send a stream of mouse
     events which receive no replies, this packetization may
     cause significant delays.   Therefore, TCP provides a boolean
     option, TCP_NODELAY (from <netinet/tcp.h>, to defeat this

algorithm. The option level for the setsockopt call is the
protocol number for TCP, available from getprotobyname(3N).

Options at the IP transport level may be used with TCP; see
ip(4P). Incoming connection requests that are source-routed
are noted, and the reverse source route is used in respond-
ing.

DIAGNOSTICS
    A socket operation may fail with one of the following errors
    returned:

    [EISCONN]   when trying to establish a connection on
                       a socket which already has one;

    [ENOBUFS]   when the system runs out of memory for
                       an internal data structure;

    [ETIMEDOUT]   when a connection was dropped due to
                       excessive retransmissions;

    [ECONNRESET] when the remote peer forces the connec-
                       tion to be closed;

    [ECONNREFUSED]   when the remote peer actively refuses
                       connection establishment (usually
                       because no process is listening to the
                       port);

    [EADDRINUSE] when an attempt is made to create a
                       socket with a port which has already
                       been allocated;

    [EADDRNOTAVAIL]   when an attempt is made to create a
                       socket with a network address for which
                       no network interface exists.

SEE ALSO
    getsockopt(2), socket(2), intro(4N), inet(4F), ip(4P)

NAME
     tm - TM-11/TE-10 magtape interface

SYNOPSIS
     /sys/conf/SYSTEM:
        # Setting AVIVTM configures the TM driver for the AVIV 800/1600/6250
        # controller. For more details, see /sys/pdpuba/tm.c.
        NTM tm_drives # TM11
        AVIVTM YES   # AVIV 800/1600/6250 controller

     /etc/dtab:
        #Name   Unit# Addr   Vector Br Handler(s)   # Comments
        tm ?   172520 224   5 tmintr   # tm11 driver

     major device number(s):
        raw: 7
        block: 1
     minor device encoding:
        bits 0003 specify TS drive
        bit 0004 specifies no-rewind operation
        bits 0030 specify recording density:
            0000: 800BPI
            0010: 1600BPI (AVIVTM and some other controllers)
            0020: 6250BPI (AVIVTM only)

DESCRIPTION
     The tm-11/te-10 combination provides a standard tape drive
     interface as described in mtio(4). The standard DEC tm-11
     operates only at 800 bpi.   Other controllers of this type
     may also allow operation at 1600 bpi, under software control
     or by switching manually.

FILES
     /dev/MAKEDEV   script to create special files
     /dev/MAKEDEV.local   script to localize special files

SEE ALSO
     mt(1), tar(1), tp(1), mtio(4), ht(4), ts(4), mt(4), dtab(5),
     autoconfig(8)

DIAGNOSTICS
     te%d: no write ring. An attempt was made to write on the
     tape drive when no write ring was present; this message is
     written on the terminal of the user who tried to access the
     tape.

     te%d: not online. An attempt was made to access the tape
     while it was offline; this message is written on the termi-
     nal of the user who tried to access the tape.

     te%d: can't change density in mid-tape. An attempt was made
     to write on a tape at a different density than is already

recorded on the tape. This message is written on the termi-
nal of the user who tried to switch the density.

te%d: hard error bn%d er=%b.   A tape error occurred at
block bn; the tm error register is printed in octal with the
bits symbolically decoded. Any error is fatal on non-raw
tape; when possible the driver will have retried the opera-
tion which failed several times before reporting the error.

te%d: lost interrupt. A tape operation did not complete
within a reasonable time, most likely because the tape was
taken off-line during rewind or lost vacuum. The controller
should, but does not, give an interrupt in these cases. The
device will be made available again after this message, but
any current open reference to the device will return an
error as the operation in progress aborts.

BUGS
     If any non-data error is encountered on non-raw tape, it
     refuses to do anything more until closed.

NAME
     tmscp - DEC TMSCP magtape interface

SYNOPSIS
     /sys/conf/SYSTEM:

          NTMSCP   1   # TMSCP controllers
          NTMS   1   # TMSCP drives
          TMSCP_DEBUG NO   # debugging code in in TMSCP drive (EXPENSIVE)

     /etc/dtab:
        #Name   Unit# Addr   Vector Br Handler(s) # Comments
        tms ?   174500 260   5   tmsintr   # tmscp driver
        tms ?   164334 0   5   tmsintr   # alternate

     major device number(s):

          raw: 23
          block: 12

     minor device encoding:

     bit: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

                C   C   X   D   D   N   U   U

        C = Controller # (max of 4 controllers)
        D = Density
        N = Norewind on close
        U = Unit (drive) number (max of 4 drives per controller)

DESCRIPTION
     Tape controllers compatible with the DEC Tape Mass Storage
     Control Protocol (TMSCP) architecture such as the TU81 and
     the TK50 provide a standard tape drive interface as
     described in mtio(4). The controller communicates with the
     host through a packet oriented protocol. Consult the file
     <pdp/tmscp.h> for a detailed description of this protocol.

FILES
     /dev/MAKEDEV script to create special files
     /dev/MAKEDEV.local script to localize special files

     The vector specified in /etc/dtab may be given as an expli-
     cit value in which case autoconfig(8) will attempt to allo-
     cate the specified vector. The vector may also be (if the
     system is at revision level 100 or higher) given as 0 - in
     which case autoconfig(8) will request the kernel to allocate
     the next available vector (beginning at 01000 and proceeding
     downward) which autoconfig(8) will initialize.

Multiple drives on a single controller are supported. Mul-
tiple controllers are also supported.

Density selection is specified by bits 3 and 4 in the minor
device number. A value of 0 requests the lowest density of
a drive and a value of 3 the highest density. Values of 1
and 2 are essentially equivalent (because because TMSCP only
defines values for three densities) and request the middle
density (for tri-density tape drives).

DIAGNOSTICS

tms: !drives. Not enough drives were declared when the ker-
nel was built. The NTMS parameter in the kernel config file
needs to be increased by at least one.

tms%d stepN init failed: sa %x. Step N of the 4 step ini-
tialization sequence has failed.

tms%d: random intr. An unexpected interrupt was received.
This is indicative of some other device using the same
interrupt vector as the TMSCP controller.   The interrupt is
ignored.

tms%d Ver %d Mod %d. The version and model number of the
controller are displayed when the controller is initialized.
This is an information message and not an error.

tms%d: state %d. The controller state is not one of the 4
initialization states or the RUN state. This indicates a
serious problem in the driver and possibly the hardware.

tms%d: fatal error %x. The controller detected a ``fatal
error'' in the status returned to the host. The contents of
the status register are displayed.

tms%d init fail. The controller failed to initialize.
Indicative of a hardware problem.

tms%d,%d flush fail. The cache failed to flush during a
close operation. Data loss is possible in this case.

tms%d,%d: sa %x state %d.   A fatal error.   The controller
will be reset in an attempt to resume operation.

tms ioctl. An invalid internal ioctl function call has been
made. This is a driver bug.

tms%d,%d inv end. An invalid end (completion) code has been
detected.   A drive has returned 0 as the opcode originally
issued. This is a hardware problem.

      tms%d,%d bad rsp: %x. An unrecognized response has been
      received.   This is a driver bug.

      tms%d,%d cache lost The cache on the drive has been lost.
      Data loss is likely. Usually due to a hardware problem.

      The following error is produced when a TMSCP error log
      datagram is received:

      tms%d,%d dgram fmt=%x evt=%x grp=%x flg=%x pos=%D.

      Earlier versions of the driver consumed a noticeable amount
      of kernel D-space decoding and pretty printing more detailed
      information. This has been removed in favor of a shorter
      message. In the future an error log daemon will be written
      and the datagrams from the MSCP and TMSCP drivers passed to
      it for analysis.

SEE ALSO
      mt(1), tar(1), tp(1), mtio(4), tm(4), ts(4), ut(4),
      dmesg(8), dtab(5), autoconfig(8)

BUGS
      If any non-data error is encountered on non-raw tape, it
      refuses to do anything more until closed.

      On quad-density tape drives (the Kennedy 9662 for example)
      the middle density of 3200bpi is not host selectable (it can
      be manually selected from the drive's front panel) because
      TMSCP only defines 800, 1600 and 6250bpi.

NAME
     ts - TS-11 magtape interface

SYNOPSIS
     /sys/conf/SYSTEM:
        NTS   ts_drives   # TS11

     /etc/dtab:
        #Name   Unit# Addr   Vector Br Handler(s)   # Comments
        ts ?   172520 224   5 tsintr   # ts11 driver

     major device number(s):
        raw: 8
        block: 2
     minor device encoding:
        bits 0003 specify TS drive
        bit 0004 specifies no-rewind operation
        bit 0010 ignored

DESCRIPTION
     The ts-11 combination provides a standard tape drive inter-
     face as described in mtio(4). The ts-11 operates only at
     1600 bpi, and only one transport is possible per controller.

FILES
     /dev/MAKEDEV   script to create special files
     /dev/MAKEDEV.local   script to localize special files

SEE ALSO
     mt(1), tar(1), tp(1), mtio(4), ht(4), tm(4), dtab(5), auto-
     config(8)

DIAGNOSTICS
     ts%d: no write ring. An attempt was made to write on the
     tape drive when no write ring was present; this message is
     written on the terminal of the user who tried to access the
     tape.

     ts%d: not online. An attempt was made to access the tape
     while it was offline; this message is written on the termi-
     nal of the user who tried to access the tape.

     ts%d: hard error bn%d xs0=%b xs1=%b xs2=%b xs3=%b. A hard
     error occurred on the tape at block bn; status registers 0-3
     are printed in octal and symbolically decoded as bits.

     ts%d: addr mod 4 != 0. The address of a TS-11 command
     packet was not on an even longword boundary.

BUGS
     If any non-data error is encountered on non-raw tape, it
     refuses to do anything more until closed.

Printed 6/17/012 January 28, 1988   1

The device lives at the same address as a tm-11 tm(4).

NAME
     tty - general terminal interface

SYNOPSIS
     #include <sgtty.h>

DESCRIPTION
     This section describes both a particular special file
     /dev/tty and the terminal drivers used for conversational
     computing.

     Line disciplines.

     The system provides different line disciplines for control-
     ling communications lines. In this version of the system
     there are two disciplines available for use with terminals:

     old   The old (Version 7) terminal driver. This is some-
             times used when using the standard shell sh(1).

     new   The standard Berkeley terminal driver, with features
             for job control; this must be used when using
             csh(1).

     Line discipline switching is accomplished with the TIOCSETD
     ioctl:

        int ldisc = LDISC;
        ioctl(f, TIOCSETD, &ldisc);

     where LDISC is OTTYDISC for the standard tty driver and
     NTTYDISC for the ``new'' driver. The standard (currently
     old) tty driver is discipline 0 by convention. Other dis-
     ciplines may exist for special purposes, such as use of com-
     munications lines for network connections. The current line
     discipline can be obtained with the TIOCGETD ioctl. Pending
     input is discarded when the line discipline is changed.

     All of the low-speed asynchronous communications ports can
     use any of the available line disciplines, no matter what
     hardware is involved. The remainder of this section
     discusses the "old" and "new" disciplines.

     The control terminal.

     When a terminal file is opened, it causes the process to
     wait until a connection is established. In practice, user
     programs seldom open these files; they are opened by
     getty(8) or rlogind(8C) and become a user's standard input
     and output file.

If a process which has no control terminal opens a terminal
file, then that terminal file becomes the control terminal
for that process. The control terminal is thereafter inher-
ited by a child process during a fork(2), even if the con-
trol terminal is closed.

The file /dev/tty is, in each process, a synonym for a con-
trol terminal associated with that process. It is useful
for programs that wish to be sure of writing messages on the
terminal no matter how output has been redirected. It can
also be used for programs that demand a file name for out-
put, when typed output is desired and it is tiresome to find
out which terminal is currently in use.

A process can remove the association it has with its con-
trolling terminal by opening the file /dev/tty and issuing
an

    ioctl(f, TIOCNOTTY, 0);

This is often desirable in server processes.

Process groups.

Command processors such as csh(1) can arbitrate the terminal
between different jobs by placing related jobs in a single
process group and associating this process group with the
terminal.   A terminal's associated process group may be set
using the TIOCSPGRP ioctl(2):

    ioctl(fildes, TIOCSPGRP, &pgrp);

or examined using TIOCGPGRP, which returns the current pro-
cess group in pgrp. The new terminal driver aids in this
arbitration by restricting access to the terminal by
processes which are not in the current process group; see
Job access control below.

Modes.

The terminal drivers have three major modes, characterized
by the amount of processing on the input and output charac-
ters:

cooked   The normal mode. In this mode lines of input are
          collected and input editing is done. The edited
          line is made available when it is completed by a
          newline, or when the t_brkc character (normally
          undefined) or t_eofc character (normally an EOT,
          control-D, hereafter ^D) is entered. A carriage
          return is usually made synonymous with newline in
          this mode, and replaced with a newline whenever it

is typed. All driver functions (input editing,
interrupt generation, output processing such as
tab expansion, etc.) are available in this mode.

CBREAK   This mode eliminates the character, word, and line
editing input facilities, making the input charac-
ter available to the user program as it is typed.
Flow control, literal-next and interrupt process-
ing are still done in this mode. Output process-
ing is done.

RAW   This mode eliminates all input processing and
makes all input characters available as they are
typed; no output processing is done either.

The style of input processing can also be very different
when the terminal is put in non-blocking I/O mode; see the
FNDELAY flag described in fcntl(2). In this case a read(2)
from the control terminal will never block, but rather
return an error indication (EWOULDBLOCK) if there is no
input available.

A process may also request that a SIGIO signal be sent it
whenever input is present and also whenever output queues
fall below the low-water mark. To enable this mode the
FASYNC flag should be set using fcntl(2).

Input editing.

A UNIX terminal ordinarily operates in full-duplex mode.
Characters may be typed at any time, even while output is
occurring, and are only lost when the system's character
input buffers become completely choked, which is rare, or
when the user has accumulated the maximum allowed number of
input characters that have not yet been read by some pro-
gram. Currently this limit is 256 characters. In RAW mode,
the terminal driver throws away all input and output without
notice when the limit is reached. In CBREAK or cooked mode
it refuses to accept any further input and, if in the new
line discipline, rings the terminal bell.

Input characters are normally accepted in either even or odd
parity with the parity bit being stripped off before the
character is given to the program. By clearing either the
EVEN or ODD bit in the flags word it is possible to have
input characters with that parity discarded (see the Summary
below.)

In all of the line disciplines, it is possible to simulate
terminal input using the TIOCSTI ioctl, which takes, as its
third argument, the address of a character. The system pre-
tends that this character was typed on the argument

terminal, which must be the control terminal except for the
super-user (this call is not in standard version 7 UNIX).

Input characters are normally echoed by putting them in an
output queue as they arrive. This may be disabled by clear-
ing the ECHO bit in the flags word using the stty(3C) call
or the TIOCSETN or TIOCSETP ioctls (see the Summary below).

In cooked mode, terminal input is processed in units of
lines. A program attempting to read will normally be
suspended until an entire line has been received (but see
the description of SIGTTIN in Job access control and of
FIONREAD in Summary, both below.) No matter how many charac-
ters are requested in the read call, at most one line will
be returned. It is not, however, necessary to read a whole
line at once; any number of characters may be requested in a
read, even one, without losing information.

During input, line editing is normally done, with the erase
character sg_erase (by default, DELETE) logically erasing
the last character typed and the sg_kill character (default,
^U: control-U) logically erasing the entire current input
line. These characters never erase beyond the beginning of
the current input line or an eof.

The drivers normally treat either a carriage return or a
newline character as terminating an input line, replacing
the return with a newline and echoing a return and a line
feed. If the CRMOD bit is cleared in the local mode word
then the processing for carriage return is disabled, and it
is simply echoed as a return, and does not terminate cooked
mode input.

In the new driver there is a literal-next character (nor-
mally ^V) which can be typed in both cooked and CBREAK mode
preceding any character to prevent its special meaning to
the terminal handler.

The new terminal driver also provides two other editing
characters in normal mode. The word-erase character, nor-
mally ^W, erases the preceding word, but not any spaces
before it. For the purposes of ^W, a word is defined as a
sequence of non-blank characters, with tabs counted as
blanks. Finally, the reprint character, normally ^R,
retypes the pending input beginning on a new line. Retyping
occurs automatically in cooked mode if characters which
would normally be erased from the screen are fouled by pro-
gram output.

Input echoing and redisplay

The terminal driver has several modes for handling the echo-
ing of terminal input, controlled by bits in a local mode
word.

Hardcopy terminals. When a hardcopy terminal is in use, the
LPRTERA bit is normally set in the local mode word. Charac-
ters which are logically erased are then printed out back-
wards preceded by `\' and followed by `/' in this mode.

CRT terminals. When a CRT terminal is in use, the LCRTBS bit
is normally set in the local mode word. The terminal driver
then echoes the proper number of erase characters when input
is erased; in the normal case where the erase character is a
^H this causes the cursor of the terminal to back up to
where it was before the logically erased character was
typed. If the input has become fouled due to interspersed
asynchronous output, the input is automatically retyped.

Erasing characters from a CRT. When a CRT terminal is in
use, the LCRTERA bit may be set to cause input to be erased
from the screen with a "backspace-space-backspace" sequence
when character or word deleting sequences are used. A
LCRTKIL bit may be set as well, causing the input to be
erased in this manner on line kill sequences as well.

Echoing of control characters. If the LCTLECH bit is set in
the local state word, then non-printing (control) characters
are normally echoed as ^X (for some X) rather than being
echoed unmodified; delete is echoed as ^?.

The normal modes for use on CRT terminals are speed depen-
dent. At speeds less than 1200 baud, the LCRTERA and
LCRTKILL processing is painfully slow, and stty(1) normally
just sets LCRTBS and LCTLECH; at speeds of 1200 baud or
greater all of these bits are normally set. Stty(1) summar-
izes these option settings and the use of the new terminal
driver as "newcrt."

Output processing.

When one or more characters are written, they are actually
transmitted to the terminal as soon as previously-written
characters have finished typing. (As noted above, input
characters are normally echoed by putting them in the output
queue as they arrive.) When a process produces characters
more rapidly than they can be typed, it will be suspended
when its output queue exceeds some limit.   When the queue
has drained down to some threshold the program is resumed.
Even parity is normally generated on output. The EOT char-
acter is not transmitted in cooked mode to prevent terminals
that respond to it from hanging up; programs using RAW or
CBREAK mode should be careful.

The terminal drivers provide necessary processing for cooked
and CBREAK mode output including parity generation. The
driver will also optionally expand tabs into spaces, where
the tab stops are assumed to be set every eight columns, and
optionally convert newlines to carriage returns followed by
newline. These functions are controlled by bits in the tty
flags word; see Summary below.

Finally, in the new terminal driver, there is a output flush
character, normally ^O, which sets the LFLUSHO bit in the
local mode word, causing subsequent output to be flushed
until it is cleared by a program or more input is typed.
This character has effect in both cooked and CBREAK modes
and causes pending input to be retyped if there is any pend-
ing input. An ioctl to flush the characters in the input or
output queues, TIOCFLUSH, is also available.

Flow control.

There are two characters (the stop character, normally ^S,
and the start character, normally ^Q) which cause output to
be suspended and resumed respectively. Extra stop charac-
ters typed when output is already stopped have no effect,
unless the start and stop characters are made the same, in
which case output resumes.

A bit in the flags word may be set to put the terminal into
TANDEM mode. In this mode the system produces a stop char-
acter (default ^S) when the input queue is in danger of
overflowing, and a start character (default ^Q) when the
input has drained sufficiently. This mode is useful when
the terminal is actually another machine that obeys those
conventions.

If software flow control is not sufficient (it begins fail-
ing above 4800 bits per second on slow systems) RTS/CTS flow
control is now available. Hardware flow control causes RTS
to be dropped when the remote system (modem) should cease
sending and to be raised when additional data can be
accepted.   If the remote system drops CTS then the local
host ceases sending data until the CTS is raised.   Hardware
flow control is enabled via the TIOCLBIS (or TIOCLBIC or
TIOCLSET) function and the bit LRTSCTS.

Line control and breaks.

There are several ioctl calls available to control the state
of the terminal line. The TIOCSBRK ioctl will set the break
bit in the hardware interface causing a break condition to
exist; this can be cleared (usually after a delay with
sleep(3)) by TIOCCBRK. Break conditions in the input are
reflected as a null character in RAW mode or as the

interrupt character in cooked or CBREAK mode. The TIOCCDTR
ioctl will clear the data terminal ready condition; it can
be set again by TIOCSDTR.

When the carrier signal from the dataset drops (usually
because the user has hung up his terminal) a SIGHUP hangup
signal is sent to the processes in the distinguished process
group of the terminal; this usually causes them to ter-
minate. The SIGHUP can be suppressed by setting the LNOHANG
bit in the local state word of the driver. Access to the
terminal by other processes is then normally revoked, so any
further reads will fail, and programs that read a terminal
and test for end-of-file on their input will terminate
appropriately.

It is possible to ask that the phone line be hung up on the
last close with the TIOCHPCL ioctl; this is normally done on
the outgoing lines and dialups.

Interrupt characters.

There are several characters that generate interrupts in
cooked and CBREAK mode; all are sent to the processes in the
control group of the terminal, as if a TIOCGPGRP ioctl were
done to get the process group and then a killpg(2) system
call were done, except that these characters also flush
pending input and output when typed at a terminal ('a`'la
TIOCFLUSH). The characters shown here are the defaults; the
field names in the structures (given below) are also shown.
The characters may be changed.

^C   t_intrc (ETX) generates a SIGINT signal. This is the
   normal way to stop a process which is no longer
   interesting, or to regain control in an interactive
   program.

^\   t_quitc (FS) generates a SIGQUIT signal. This is used
   to cause a program to terminate and produce a core
   image, if possible, in the file core in the current
   directory.

^Z   t_suspc (EM) generates a SIGTSTP signal, which is used
   to suspend the current process group.

^Y   t_dsuspc (SUB) generates a SIGTSTP signal as ^Z does,
   but the signal is sent when a program attempts to read
   the ^Y, rather than when it is typed.

Job access control.

When using the new terminal driver, if a process which is
not in the distinguished process group of its control

terminal attempts to read from that terminal its process
group is sent a SIGTTIN signal. This signal normally causes
the members of that process group to stop. If, however, the
process is ignoring SIGTTIN, has SIGTTIN blocked, or is in
the middle of process creation using vfork(2)), the read
will return -1 and set errno to EIO.

When using the new terminal driver with the LTOSTOP bit set
in the local modes, a process is prohibited from writing on
its control terminal if it is not in the distinguished pro-
cess group for that terminal. Processes which are holding
or ignoring SIGTTOU signals or which are in the middle of a
vfork(2) are excepted and allowed to produce output.
Terminal/window sizes. In order to accommodate terminals and
workstations with variable-sized windows, the terminal
driver provides a mechanism for obtaining and setting the
current terminal size. The driver does not use this infor-
mation internally, but only stores it and provides a uniform
access mechanism. When the size is changed, a SIGWINCH sig-
nal is sent to the terminal's process group so that
knowledgeable programs may detect size changes. This facil-
ity was added in 4.3BSD and is not available in earlier ver-
sions of the system.

Summary of modes.

Unfortunately, due to the evolution of the terminal driver,
there are 4 different structures which contain various por-
tions of the driver data.   The first of these (sgttyb) con-
tains that part of the information largely common between
version 6 and version 7 UNIX systems. The second contains
additional control characters added in version 7.   The third
is a word of local state added in 4BSD, and the fourth is
another structure of special characters added for the new
driver. In the future a single structure may be made avail-
able to programs which need to access all this information;
most programs need not concern themselves with all this
state.

Basic modes: sgtty.

The basic ioctls use the structure defined in <sgtty.h>:

```
struct sgttyb {
    char sg_ispeed;
    char sg_ospeed;
    char sg_erase;
    char sg_kill;
    shortsg_flags;
};
```

The sg_ispeed and sg_ospeed fields describe the input and
output speeds of the device according to the following
table, which corresponds to the DEC DH-11 interface. If
other hardware is used, impossible speed changes are
ignored. Symbolic values in the table are as defined in
<sgtty.h>.

```
B0      0       (hang up dataphone)
B50     1        50 baud
B75     2        75 baud
B110    3       110 baud
B134    4       134.5 baud
B150    5       150 baud
B200    6       200 baud
B300    7       300 baud
B600    8       600 baud
B1200   9      1200 baud
B1800   10     1800 baud
B2400   11     2400 baud
B4800   12     4800 baud
B9600   13     9600 baud
EXTA    14      External A
EXTB    15      External B
```

Code conversion and line control required for IBM 2741's
(134.5 baud) must be implemented by the user's program. The
half-duplex line discipline required for the 202 dataset
(1200 baud) is not supplied; full-duplex 212 datasets work
fine.

The sg_erase and sg_kill fields of the argument structure
specify the erase and kill characters respectively.
(Defaults are DELETE and ^U.)

The sg_flags field of the argument structure contains
several bits that determine the system's treatment of the
terminal:

```
XTABS  0004000 Expand tabs on output
EVENP  0000200 Even parity allowed on input
ODDP   0000100 Odd parity allowed on input
RAW    0000040 Raw mode: wake up on all characters, 8-bit interface
CRMOD  0000020 Map CR into LF; output LF as CR-LF
ECHO   0000010 Echo (full duplex)
CBREAK 0000002 Return each character as soon as typed
TANDEM 0000001 Automatic inbound xon/xoff flow control
```

XTABS, causes tabs to be replaced by the appropriate number
of spaces on output.

The flags for even and odd parity control parity checking on
input and generation on output in cooked and CBREAK mode

(unless LPASS8 is enabled, see below). Even parity is gen-
erated on output unless ODDP is set and EVENP is clear, in
which case odd parity is generated. Input characters with
the wrong parity, as determined by EVENP and ODDP, are
ignored in cooked and CBREAK mode.

RAW disables all processing save output flushing with
LFLUSHO; full 8 bits of input are given as soon as it is
available; all 8 bits are passed on output. A break condi-
tion in the input is reported as a null character. If the
input queue overflows in raw mode all data in the input and
output queues are discarded; this applies to both new and
old drivers.

CRMOD causes input carriage returns to be turned into new-
lines, and output and echoed new-lines to be output as a
carriage return followed by a line feed.

CBREAK is a sort of half-cooked (rare?) mode. Programs can
read each character as soon as typed, instead of waiting for
a full line; all processing is done except the input edit-
ing: character and word erase and line kill, input reprint,
and the special treatment of \ and EOT are disabled.

TANDEM mode causes the system to produce a stop character
(default ^S) whenever the input queue is in danger of over-
flowing, and a start character (default ^Q) when the input
queue has drained sufficiently. It is useful for flow con-
trol when the `terminal' is really another computer which
understands the conventions.

Note: The same ``stop'' and ``start'' characters are used
for both directions of flow control; the t_stopc character
is accepted on input as the character that stops output and
is produced on output as the character to stop input, and
the t_startc character is accepted on input as the character
that restarts output and is produced on output as the char-
acter to restart input.

Basic ioctls

A large number of ioctl(2) calls apply to terminals. Some
have the general form:

#include <sgtty.h>

ioctl(fildes, code, arg)
struct sgttyb *arg;

The applicable codes are:

TIOCGETP   Fetch the basic parameters associated with

the terminal, and store in the pointed-to
sgttyb structure.

TIOCSETP   Set the parameters according to the pointed-
                to sgttyb structure. The interface delays
                until output is quiescent, then throws away
                any unread characters, before changing the
                modes.

TIOCSETN   Set the parameters like TIOCSETP but do not
                delay or flush input. Input is not
                preserved, however, when changing to or from
                RAW.

With the following codes arg is ignored.

TIOCEXCL   Set "exclusive-use" mode: no further opens
                are permitted until the file has been closed.

TIOCNXCL   Turn off "exclusive-use" mode.

TIOCHPCL   When the file is closed for the last time,
                hang up the terminal. This is useful when
                the line is associated with an ACU used to
                place outgoing calls.

With the following codes arg is a pointer to an int.

TIOCGETD   arg is a pointer to an int into which is
                placed the current line discipline number.

TIOCSETD   arg is a pointer to an int whose value
                becomes the current line discipline number.

TIOCFLUSH   If the int pointed to by arg has a zero
                value, all characters waiting in input or
                output queues are flushed.   Otherwise, the
                value of the int is for the FREAD and FWRITE
                bits defined in <sys/file.h>; if the FREAD
                bit is set, all characters waiting in input
                queues are flushed, and if the FWRITE bit is
                set, all characters waiting in output queues
                are flushed.

The remaining calls are not available in vanilla version 7
UNIX. In cases where arguments are required, they are
described; arg should otherwise be given as 0.

TIOCSTI   the argument points to a character which the
                system pretends had been typed on the termi-
                nal.

TIOCSBRK   the break bit is set in the terminal.

TIOCCBRK   the break bit is cleared.

TIOCSDTR   data terminal ready is set.

TIOCCDTR   data terminal ready is cleared.

TIOCSTOP   output is stopped as if the ``stop'' charac-
                 ter had been typed.

TIOCSTART   output is restarted as if the ``start'' char-
                 acter had been typed.

TIOCGPGRP   arg is a pointer to an int into which is
                 placed the process group ID of the process
                 group for which this terminal is the control
                 terminal.

TIOCSPGRP   arg is a pointer to an int which is the value
                 to which the process group ID for this termi-
                 nal will be set.

TIOCOUTQ   returns in the int pointed to by arg the
                 number of characters queued for output to the
                 terminal.

FIONREAD   returns in the long pointed to by arg the
                 number of characters immediately readable
                 from the argument descriptor. This works for
                 files, pipes, and terminals.

Tchars

The second structure associated with each terminal specifies
characters that are special in both the old and new terminal
interfaces: The following structure is defined in
<sys/ioctl.h>, which is automatically included in <sgtty.h>:

```
struct tchars {
   char t_intrc;  /* interrupt */
   char t_quitc;  /* quit */
   char t_startc; /* start output */
   char t_stopc;  /* stop output */
   char t_eofc;   /* end-of-file */
   char t_brkc;   /* input delimiter (like nl) */
};
```

The default values for these characters are ^C, ^\, ^Q, ^S,
^D, and -1. A character value of -1 eliminates the effect
of that character. The t_brkc character, by default -1,
acts like a new-line in that it terminates a `line,' is

echoed, and is passed to the program. The `stop' and
`start' characters may be the same, to produce a toggle
effect. It is probably counterproductive to make other spe-
cial characters (including erase and kill) identical. The
applicable ioctl calls are:

TIOCGETC   Get the special characters and put them in the
             specified structure.

TIOCSETC   Set the special characters to those given in the
             structure.

Local mode

The third structure associated with each terminal is a local
mode word. The bits of the local mode word are:

LCRTBS   000001   Backspace on erase rather than echoing erase
LPRTERA   000002   Printing terminal erase mode
LCRTERA   000004   Erase character echoes as backspace-space-backspace
LMDMBUF   000020   Stop/start output when carrier drops
LLITOUT   000040   Suppress output translations
LTOSTOP   000100   Send SIGTTOU for background output
LFLUSHO   000200   Output is being flushed
LNOHANG   000400   Don't send hangup when carrier drops
LRTSCTS   001000   RTS/CTS flow control
LCRTKIL   002000   BS-space-BS erase entire line on line kill
LPASS8   004000   Pass all 8 bits through on input, in any mode
LCTLECH   010000   Echo input control chars as ^X, delete as ^?
LPENDIN   020000   Retype pending input at next read or input character
LDECCTQ   040000   Only ^Q restarts output after ^S, like DEC systems
LNOFLSH   100000   Inhibit flushing of pending I/O when an interrupt
                    character is typed.

The applicable ioctl functions are:

TIOCLBIS   arg is a pointer to an int whose value is a
                mask containing the bits to be set in the
                local mode word.

TIOCLBIC   arg is a pointer to an int whose value is a
                mask containing the bits to be cleared in the
                local mode word.

TIOCLSET   arg is a pointer to an int whose value is
                stored in the local mode word.

TIOCLGET   arg is a pointer to an int into which the
                current local mode word is placed.

Local special chars

The final control structure associated with each terminal is
the ltchars structure which defines control characters for
the new terminal driver. Its structure is:

```
struct ltchars {
   char t_suspc;  /* stop process signal */
   char t_dsuspc; /* delayed stop process signal */
   char t_rprntc; /* reprint line */
   char t_flushc; /* flush output (toggles) */
   char t_werasc; /* word erase */
   char t_lnextc; /* literal next character */
};
```

The default values for these characters are ^Z, ^Y, ^R, ^O,
^W, and ^V. A value of -1 disables the character.

The applicable ioctl functions are:

TIOCSLTC   arg is a pointer to an ltchars structure which
        defines the new local special characters.

TIOCGLTC   arg is a pointer to an ltchars structure into
        which is placed the current set of local special
        characters.

Window/terminal sizes

Each terminal has provision for storage of the current ter-
minal or window size in a winsize structure, with format:

```
struct winsize {
   unsigned short    ws_row;     /* rows, in characters */
   unsigned short    ws_col;     /* columns, in characters */
   unsigned short    ws_xpixel;     /* horizontal size, pixels */
   unsigned short    ws_ypixel;     /* vertical size, pixels */
};
```

A value of 0 in any field is interpreted as ``undefined;''
the entire structure is zeroed on final close.

The applicable ioctl functions are:

TIOCGWINSZ
  arg is a pointer to a struct winsize into which will be
  placed the current terminal or window size information.

TIOCSWINSZ
  arg is a pointer to a struct winsize which will be used
  to set the current terminal or window size information.
  If the new information is different than the old infor-
  mation, a SIGWINCH signal will be sent to the
  terminal's process group.

FILES
    /dev/tty
    /dev/tty*
    /dev/console

SEE ALSO
    csh(1), stty(1), tset(1), ioctl(2), sigvec(2), stty(3C),
    getty(8)

NAME
     udp - Internet User Datagram Protocol

SYNOPSIS
     #include <sys/socket.h>
     #include <netinet/in.h>

     s = socket(AF_INET, SOCK_DGRAM, 0);

DESCRIPTION
     UDP is a simple, unreliable datagram protocol which is used
     to support the SOCK_DGRAM abstraction for the Internet pro-
     tocol family. UDP sockets are connectionless, and are nor-
     mally used with the sendto and recvfrom calls, though the
     connect(2) call may also be used to fix the destination for
     future packets (in which case the recv(2) or read(2) and
     send(2) or write(2) system calls may be used).

     UDP address formats are identical to those used by TCP. In
     particular UDP provides a port identifier in addition to the
     normal Internet address format. Note that the UDP port
     space is separate from the TCP port space (i.e. a UDP port
     may not be "connected" to a TCP port). In addition broad-
     cast packets may be sent (assuming the underlying network
     supports this) by using a reserved "broadcast address"; this
     address is network interface dependent.

     Options at the IP transport level may be used with UDP; see
     ip(4P).

DIAGNOSTICS
     A socket operation may fail with one of the following errors
     returned:

     [EISCONN]   when trying to establish a connection on a
                     socket which already has one, or when trying
                     to send a datagram with the destination
                     address specified and the socket is already
                     connected;

     [ENOTCONN]   when trying to send a datagram, but no desti-
                     nation address is specified, and the socket
                     hasn't been connected;

     [ENOBUFS]   when the system runs out of memory for an
                     internal data structure;

     [EADDRINUSE]   when an attempt is made to create a socket
                     with a port which has already been allocated;

     [EADDRNOTAVAIL]
                     when an attempt is made to create a socket

Printed 6/17/012   May 16, 1986   1

with a network address for which no network
interface exists.

SEE ALSO
getsockopt(2), recv(2), send(2), socket(2), intro(4N),
inet(4F), ip(4P)

NAME
     vv - Proteon proNET 10 Megabit ring

SYNOPSIS
     /sys/conf/SYSTEM:
        NVV vv_controllers # V2LNI (Pronet)

DESCRIPTION
     The vv interface provides access to a 10 Mb/s Proteon proNET
     ring network.

     The network address of the interface must be specified with
     an an SIOCSIFADDR ioctl before data can be transmitted or
     received.   It is only permissible to change the network
     address while the interface is marked "down".

     The host's hardware address is discovered by putting the
     interface in digital loopback mode (not joining the ring)
     and sending a broadcast packet from which the hardware
     address is extracted.

     Transmit timeouts are detected through use of a watchdog
     routine. Lost input interrupts are checked for when packets
     are sent out.

     If the installation is running CTL boards which use the old
     broadcast address of 0 instead of the new address of 0xff,
     the define OLD_BROADCAST should be specified in the driver.

     The driver can use ``trailer'' encapsulation to minimize
     copying data on input and output. This may be disabled, on
     a per-interface basis, by setting the IFF_NOTRAILERS flag
     with an SIOCSIFFLAGS ioctl.

DIAGNOSTICS
     vv%d: host %d. The software announces the host address
     discovered during autoconfiguration.

     vv%d: can't initialize. The software was unable to discover
     the address of this interface, so it deemed "dead" will not
     be enabled.

     vv%d: error vvocsr=%b. The hardware indicated an error on
     the previous transmission.

     vv%d: output timeout. The token timer has fired and the
     token will be recreated.

     vv%d: error vvicsr=%b. The hardware indicated an error in
     reading a packet off the ring.

en%d: can't handle af%d. The interface was handed a message
with addresses formatted in an unsuitable address family;
the packet was dropped.

vv%d: vs_olen=%d. The ring output routine has been handed a
message with a preposterous length. This results in an
immediate panic: vs_olen.

SEE ALSO
     intro(4N), inet(4F)

BUGS
     The encapsulation of trailer packets in the 4.2BSD version
     of this driver was incorrect (the packet type was in VAX
     byte order). As a result, the trailer encapsulation in this
     version is not compatible with the 4.2BSD VAX version.

NAME
     xp - generic SMD moving-head disk

SYNOPSIS
     /sys/conf/SYSTEM:
        NXPC   xp_controllers # Number of controllers
        NXPD   xp_drives   # RM02/03/05, RP04/05/06,
                      `   # CDC 9766, Fuji 160, etc.
        BADSECT NO   # Bad sector handling (see BUGS)

     /etc/dtab:
        #Name   Unit# Addr   Vector Br Handler(s)   # Comments
        xp ?   176700 254   5 xpintr   # xp driver

     major device number(s):
        raw: 19
        block: 10
     minor device encoding:
        bits 0007 specify partition of XP drive
        bits 0370 specify XP drive

DESCRIPTION
     The xp driver is a generic SMD storage module disk driver.
     It can be adapted to most SMD controllers although
     bootstrapping will not necessarily be directly possible.
     The drives are numbered from 0 to n on controller 0, from
     n+1 to m on controller 1, etc. The drives may have dif-
     ferent geometries.

     The xp driver is unique amoungst 2BSD drivers in its number-
     ing of drives. Other drivers (ra for example) number drives
     0 thru 7 on controller 1, 8 thru 15 on controller 2 and so
     on. xp on the other hand can have drives 0 and 1 on con-
     troller 1, drives 2, 3, 4 and 5 on controller 2 and drives
     6, 7 and 8 on controller 3. This is different from boot's
     view of the world, so if you are booting from other than
     unit 0 you may have to experiment a bit.

     Files with minor device numbers 0 through 7 refer to various
     portions of drive 0; minor devices 8 through 15 refer to
     drive 1, etc. The standard device names begin with ``xp''
     followed by the drive number and then a letter a-h for par-
     titions 0-7 respectively.   The character ? stands here for a
     drive number in the range 0-7.

     The block files access the disk via the system's normal
     buffering mechanism and may be read and written without
     regard to physical disk records. There is also a `raw'
     interface which provides for direct transmission between the
     disk and the user's read or write buffer.   A single read or
     write call results in exactly one I/O operation and there-
     fore raw I/O is considerably more efficient when many words

Printed 6/17/012 August 14, 1995   1

are transmitted. The names of the raw files conventionally
begin with an extra `r.'

In raw I/O the buffer must begin on a word (even) boundary,
and counts should be a multiple of 512 bytes (a disk sec-
tor). Likewise lseek calls should specify a multiple of 512
bytes.

DISK SUPPORT
     Disks must be labeled using either the standalone disklabel
     program on the boot tape or with the disklabel(8) program.
     The kernel no longer attempts to determine the drive type
     and geometry, instead reading this information from the
     disklabel. There are no partition tables coded into the xp
     driver, these must be placed on the drive with disklabel.

     Special files should only be created for the partitions that
     are actually used, as the overlap in these addresses could
     lead to confusion otherwise. Traditionally the xp?a parti-
     tion is normally used for the root file system, the xp?b
     partition as a swap area, and the xp?c partition for pack-
     pack copying (it maps the entire disk).

FILES
     /dev/xp[0-7][a-h]   block files
     /dev/rxp[0-7][a-h]   raw files
     /dev/MAKEDEV   script to create special files
     /dev/MAKEDEV.local   script to localize special files

SEE ALSO
     hk(4), ra(4), ram(4), rk(4), rl(4), rp(4), rx(4), si(4),
     dtab(5), autoconfig(8), newfs(8)

DIAGNOSTICS
     xp%d%c: hard error sn%d cs2=%b er1=%b. An unrecoverable
     error occurred during transfer of the specified sector of
     the specified disk partition. The contents of the two error
     registers are also printed in octal and symbolically with
     bits decoded. The error was either unrecoverable, or a
     large number of retry attempts (including offset positioning
     and drive recalibration) could not recover the error.

     xp%d: write locked. The write protect switch was set on the
     drive when a write was attempted. The write operation is
     not recoverable.

     xp%d%c: soft ecc sn%d. A recoverable ECC error occurred on
     the specified sector of the specified disk partition. This
     happens normally a few times a week. If it happens more
     frequently than this the sectors where the errors are occur-
     ring should be checked to see if certain cylinders on the
     pack, spots on the carriage of the drive or heads are

indicated.

xp%d: unknown device type 0%o. The number in the drive's
drive type register is unknown to the xp driver.

BUGS

In raw I/O read and write(2) truncate file offsets to 512-
byte block boundaries, and write scribbles on the tail of
incomplete blocks. Thus, in programs that are likely to
access raw devices, read, write and lseek(2) should always
deal in 512-byte multiples.

DEC-standard error logging should be supported.

The kernel uses partition 'h' to access the badblock infor-
mation. This should have been 'c' except that almost all of
the /etc/disktab entries (and thus existing systems) use 'h'
for this purpose. Unless you are very careful with diskla-
bel|(8) (to make certain that no data partition overlaps the
badblock area) you should probably leave BADSECT undefined
in the kernel config file.