NAME

controller, disk, tape, at, bios, esdi, aha1540, ncr810, dosfile, fatfile − controllers, disks and tapes

DESCRIPTION

The **c*n****** family of devices refer to drivers that control disks, disk like devices, and tapes. Minix contains a number of drivers for several different controllers. These controllers can have disks, cdroms and tapes attached to them. Boot Monitor variables specify which drivers are activated using the variables **c0**, **c1**, etc. The names of the devices in **/dev** that correspond with the driver for controller 0 are all named beginning with **c0**.

For each controller, the minor device numbers are organized as follows:

minor	**device**	**what?**	**obsolete**
0	d0	disk 0	hd0
1	d0p0	disk 0, partition 0	hd1
2	d0p1	disk 0, partition 1	hd2
3	d0p2	disk 0, partition 2	hd3
4	d0p3	disk 0, partition 3	hd4
5	d1	disk 1	hd5
6	d1p0	disk 1, partition 0	hd6
7	d1p1	disk 1, partition 1	hd7
8	d1p2	disk 1, partition 2	hd8
9	d1p3	disk 1, partition 3	hd9
...	...		
39	d7p3	disk 7, partition 3	hd39
64	t0n	tape 0, non-rewinding	
65	t0	tape 0, rewind on close	
66	t1n	tape 1, non-rewinding	
67	t1	tape 1, rewind on close	
...	...		
78	t7n	tape 7, non-rewinding	
79	t7	tape 7, rewind on close	
120	r0	raw access device 0	
121	r1	raw access device 1	
...	...		
127	r7	raw access device 7	
128	d0p0s0	disk 0, part 0, subpart 0	hd1a
129	d0p0s1	disk 0, part 0, subpart 1	hd1b
130	d0p0s2	disk 0, part 0, subpart 2	hd1c
131	d0p0s3	disk 0, part 0, subpart 3	hd1d
132	d0p1s0	disk 0, part 1, subpart 0	hd2a
...	...		
144	d1p0s0	disk 1, part 0, subpart 0	hd6a
...	...		
255	d7p3s3	disk 7, part 3, subpart 3	hd39d

The device names in **/dev** also name the controller, of course, so the usual place for the Minix root device, the first subpartition of the second partition of disk 0 on controller 0 is **/dev/c0d0p1s0**. Note that everything is numbered from 0! The first controller is controller 0, the first disk is disk 0, etc. So the second partition is **p1**.

The fourth column in the table above shows the disk devices names that were used by previous versions of Minix for what is now controller 0. These devices are no longer present in **/dev**.

Disks

Most disks are arrays of 512 byte sectors. The disk devices are normally block devices, which means they are block buffered by the Minix file system cache using 1024 byte blocks. The FS cache allows I/O at any byte offset, and takes care of cutting and pasting incomplete blocks together. If one creates a character device for a disk device, then I/O must be in multiples of the disk block size.

For each disk there is a device that covers the entire disk, these are named **c0d0**, **c0d1**, etc, up to **c0d7** for controller 0. If a partition table is placed in the first sector of the disk, then the disk is subdivided into regions named partitions. Up to four partitions may be defined, named **c0d0p0** to **c0d0p3** for disk 0 on controller 0. To make things interesting you can also place a partition table in the first sector of a Minix partition, which divides the partition into up to four subpartitions. Normally Minix is installed into a single partition, with the root, swap and /usr file systems in subpartitions.

If a partition is an extended partition then it contains a linked list of partition tables each of which may specify a logical partition. Up to four of these logical partitions are presented by the driver as subpartitions of the extended partition.

A sector containing a partition table starts with 446 bytes of boot code, followed by four partition table entries of 16 bytes each, and ends with the magic number 0xAA55 (little endian, so first 0x55 then 0xAA.) Partition table information is defined in <ibm/partition.h>:

```
/* Description of entry in the partition table.  */

struct part_entry {
    unsigned char bootind;         /* boot indicator 0/ACTIVE_FLAG     */
    unsigned char start_head;      /* head value for first sector      */
    unsigned char start_sec;       /* sector value + high 2 cyl bits   */
    unsigned char start_cyl;       /* low 8 cylinder bits              */
    unsigned char sysind;          /* system indicator                 */
    unsigned char last_head;       /* h/s/c for the last sector        */
    unsigned char last_sec;
    unsigned char last_cyl;
    unsigned long lowsec;          /* logical first sector             */
    unsigned long size;            /* size of partition in sectors     */
};

#define ACTIVE_FLAG     0x80   /* value for active in bootind field    */
#define NR_PARTITIONS   4      /* number of entries in table           */
#define PART_TABLE_OFF 0x1BE   /* offset of table in boot sector       */

/* Partition types (sysind). */
#define NO_PART         0x00   /* unused entry */
#define MINIX_PART      0x81   /* Minix partition type */
```

The cylinder numbers are encoded in a very strange way, bits 8 and 9 are in the high two bits of the sector number. The sector numbers count from 1, not 0! More useful are the lowsec and size fields however, they simply give the location of the partition as an absolute sector offset and length within the drive.

The partition table entry defined above is specific to IBM type disks. The device drivers use another partition entry structure to pass information on a partition. This is what <minix/partition.h> looks like:

```
struct partition {
    u64_t base;             /* byte offset to the partition start */
    u64_t size;             /* number of bytes in the partition */
    unsigned cylinders;     /* disk geometry for partitioning */
    unsigned heads;
    unsigned sectors;
};
```

The base and size fields are the byte offset and length of a partition. The geometry of the disk is also given for the benefit of partition table editors. This information can be obtained from an open disk device with the call:

ioctl(*fd***, DIOCGETP, &***entry***);**

One can change the placement of the device to the lowsec and size fields of *entry* by using the **DIOCSETP** call instead. Only the base and size fields are used for **DIOCSETP**.

The partition tables when read from disk by the driver are checked and truncated to fit within the primary partition or drive. The first sector is normally left free for the partition table.

The partition tables are read when the in-use count (opens and mounts) changes from 0 to 1. So an idle disk is automatically repartitioned on the next access. This means that DIOCSETP only has effect if the disk is in use.

Disk-like devices

Devices like a CD-ROM are treated as read-only disks, and can be accessed using disk devices. A CD-ROM usually has a block size of 2048 bytes, but the driver knows this, and allows one to read at any byte offset by reading what isn't needed into a scratch buffer.

Tapes

There are two kinds of tape devices: Non-rewinding, and rewind-on-close. The non-rewinding devices treat the tape as a series of files. The rewind-on-close devices look at the tape as a single file, and when you close such a device the tape is told to rewind. See **mt**(1), and **mtio**(4) for a description of the commands that may be sent to the tape, either from the command prompt or from a program.

There are two kinds of tape drives: Fixed and variable block size tape drives. Examples of the first kind are cartridge tapes, with a fixed 512 bytes block size. An Exabyte tape drive has a variable block size, with a minimum of 1 byte and a maximum of 245760 bytes (see the documentation of such devices.) The maximum is truncated to 32767 bytes for Minix-86 and 61440 bytes for Minix-vmd, because the driver can't move more bytes in a single request.

A read or write to a fixed block size tape must be a precise multiple of the block size, any other count gives results in an I/O error. A read from a variable block sized tape must be large enough to accept the block that is read, otherwise an I/O error will be returned. A write can be any size above the minimum, creating a block of that size. If the write count is larger than the maximum block size then more blocks are written until the count becomes zero. The last block must be larger than the minimum of course. (This minimum is often as small as 1 byte, as for the Exabyte.)

The **mt blksize** command may be used to select a fixed block size for a variable block sized tape. This will speed up I/O considerably for small block sizes. (Some systems can only use fixed mode and will write an Exabyte tape with 1024 byte blocks, which read very slow in variable mode.)

A tape is a sequence of blocks and filemarks. A tape may be opened and blocks may be read from it upto a filemark, after that all further reads return 0. After the tape is closed and reopened one can read the blocks following the filemark if using a non-rewinding device. This makes the tape look like a sequence of files.

If a tape has been written to or opened in write-only mode, then a filemark is written if the tape is closed or if a space command is issued. No extra filemark is written if the drive is instructed to write filemarks.

Raw Access Devices

Under Minix-vmd one can use the raw access devices to program a SCSI device entirely from user mode. The disk and tape devices probe for devices when opened, start disks and load tapes, but the raw access devices do nothing at all. Given an open file descriptor to any SCSI character device (not just the raw access devices) one can use the following ioctl:

```
ioctl(fd, SCIOCCMD, &scsicmd)
```

The structure whose address is passed as the third argument is defined in <sys/scsi.h> as follows:

```
struct scsicmd {
    void        *cmd;
    size_t      cmdlen;
    void        *buf;
```

```
        size_t          buflen;
        void            *sense;
        size_t          senselen;
        int             dir;
    };
```

Cmd and **cmdlen** hold the address and length of an object holding a Group 0 or Group 1 SCSI command. The next two fields describe a buffer of at most 8 kilobytes used in the data in or out phase. **Dir** is 0 if data is to be read from the device, 1 if data is written to the device. If the ioctl succeeds then 0 is returned, otherwise -1 with **errno** set to **EIO** and the request sense info returned in the buffer described by the sense and senselen fields. If the sense key is zero on error then a host adapter error occurred, this means that the device is most likely turned off or not present.

DRIVERS

By setting the Boot variables **c0** to **c3** under Minix, or **c0** to **c4** under Minix-vmd one attaches a set of disk and tape devices to a driver. See **boot**(8) for a list of boot variables that configure each of these drivers. The following drivers are available:

at

The standard IBM/AT disk driver that also supports IDE disks. This is the default driver for controller 0 on AT class machines. (Most PCs are in that class.)

bios

A disk driver that uses BIOS calls to do disk I/O. This is the default driver on anything but an AT. (Old XTs and PS/2s.) On an XT this is the best driver you can use, but on any other machine this driver may be somewhat slow, because the system has to switch out of protected mode to make a BIOS call. On a fast enough machine with a high enough setting of DMA_SECTORS (see **config**(8)) it works well enough.

esdi

A hard disk driver for use on some PS/2 models.

xt (Minix only)

A hard disk driver for IBM/XT type hard disks. Useful for old 286 based machines that have such a disk. On XTs you are better off with the **bios** driver.

aha1540

A SCSI driver for the Adaptec 1540 host adapter family, which includes the 1540, 1540A, 1540B, 1540C, 1540CF, 1640, and 1740. Also supported is the compatible BusLogic 545.

ncr810

This will eventually become a Symbios 810 SCSI driver. (Formerly owned by NCR.) KJB has read the docs on this card three times, but has still done nothing, the lazy bum.

dosfile

The "DOS file as disk" driver that is used when Minix is running under DOS. It treats a large DOS file as a Minix disk. Only primary partitions are supported, there are no subpartitions. This is the default driver when Minix is started under DOS.

fatfile

Uses a large file on a FAT file system as a disk. It needs one of the other disk drivers to do the actual I/O. This driver only knows how to interpret a FAT file system to find the file to use. With a fast native disk driver this driver is much faster than the **dosfile** driver.

FILES

/dev/c*d*	Disks devices.
/dev/c*d*p*	Partitions.
/dev/c*d*p*s*	Subpartitions.
/dev/c*t*n, /dev/c*t*	Tapes.

/dev/c*r* Raw access devices.

SEE ALSO

dd(1), **mt**(1), **eject**(1), **ioctl**(2), **int64**(3), **mtio**(4), **boot**(8), **config**(8), **monitor**(8), **part**(8), **repartition**(8).

BUGS

The subpartitioning is incompatible with the MS-DOS method of extended partitions. The latter does not map well to the sparse minor device number space.

The primary partition table is sorted by lowsec like MS-DOS does, subpartition tables are not. Just think about what happens when you delete a partition in the MS-DOS scheme.

Don't move a partition that is mounted or kept open by some process. The file system may write cached blocks to the new location.

The BIOS driver is not slow at all on a buffered disk.

Some IDE disks send an interrupt when they spin down under hardware power management. The driver acknowledges the interrupt as it is supposed to do by reading the status register. The disk then spins up again... You have to disable the spin down in the computer setup to fix the problem.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)