CTIX™ OPERATING SYSTEM MANUAL

Version C
Volume 4

This manual was prepared on a Convergent Technologies S/640 Computer System and was printed on an Apple LaserWriter II Laser Printer.

Second Edition (November 1989) 09-02265-01
Update Notice 1 (November 1990) 09-02578

TABLE OF CONTENTS: VOLUME 4

5. Miscellaneous Facilities

6. Games

7. Special Files

NAME

filehdr - file header for common object files

SYNOPSIS

#include <filehdr.h>

DESCRIPTION

Every common object file begins with a 20-byte header. The following C **struct** declaration is used:

```
struct filehdr
{
    unsigned short f_magic;     /* magic number */
    unsigned short f_nscns;     /* number of sections */
    long f_timdat;              /* time & date stamp */
    long f_symptr;              /* file ptr to symtab */
    long f_nsyms;               /* # symtab entries */
    unsigned short f_opthdr;    /* sizeof(opt hdr) */
    unsigned short f_flags;     /* flags */
};
```

f_symptr is the byte offset into the file at which the symbol table can be found. Its value can be used as the offset in *fseek*(3S) to position an I/O stream to the symbol table. The operating system optional header is always 36 bytes. The valid magic numbers are given below.

```
#define MC68KWRMAGIC    0520    /* writeable text segments */
#define MC68KROMAGIC    0521    /* readonly shareable text segments */
#define MC68KPGMAGIC    0522    /* demand paged text segments */
```

The value in *f_timdat* is obtained from the *time*(2) system call.

Flag bits currently defined are as follows:

```
#define  F_RELFLG   0000001   /* relocation entries stripped */
#define  F_EXEC     0000002   /* file is executable */
#define  F_LNNO     0000004   /* line numbers stripped */
#define  F_LSYMS    0000010   /* local symbols stripped */
#define  F_MINMAL   0000020   /* minimal object file */
#define  F_UPDATE   0000040   /* update file, ogen produced */
#define  F_SWABD    0000100   /* file is "pre-swabbed" */
#define  F_AR32W    0001000   /* non-DEC host, including Convergent
                              /* Technologies systems */
#define  F_PATCH    0002000   /* "patch" list in opt hdr */
```

The CPU type is encoded in bits 04000 and 010000. The FPU (floating-point unit) type is encoded in bits 0100000, 040000, and 020000. Macros are defined to set and extract the CPU and FPU values are as follows:

```
SETFPU(flag, value)
SETCPU(flag, value)
GETFPU(flag)
GETCPU(flag)
```

Valid values for CPU are as follows:

```
#define  F_M68010    0
#define  F_M68020    1
```

The new C compiler generates code that works on both MC68020 and MC68040-based systems. The MC68040 instruction set is a superset of the MC68020 set; therefore, MC68020 values are valid on MC68040 systems.

Valid values for FPU are as follows:

```
#define  F_NOFPU     0
#define  F_SOFT      1
#define  F_M68881    2
#define  F_SKY       4
```

SEE ALSO

time(2), fseek(3S), a.out(4).

NAME

passwd - password file

DESCRIPTION

The **/etc/passwd** file contains for each user the following information:

- login name
- encrypted password
- numerical user ID
- numerical group ID
- user name
- initial working directory
- program to use as shell

This is an ASCII file. Each field within each user entry is separated by a colon. Each user entry is separated by a new-line. If the password field is null, no password is demanded; if the shell field is null, **/bin/sh** is used.

The file contains user login information; it has general read permission and can be used, for example, to map numerical user IDs to names.

Note that if an **/etc/shadow** file exists, encrypted passwords are stored in the **/etc/shadow** file, not in **/etc/passwd**. The password field remains in **/etc/passwd** for compatibility reasons only when **/etc/shadow** exists. If the password field in **/etc/passwd** contains an **x**, the encrypted password for that login is stored in the **/etc/shadow** file. If the login does not have a password, the password field in **/etc/passwd** is empty.

If **/etc/shadow** does not exist and the login has a password, the password field in **/etc/passwd** contains the encrypted password.

The encrypted password consists of 13 characters chosen from a 64-character alphabet (., /, **0-9**, **A-Z**, **a-z**), except when the password is null, in which case the encrypted password is also null. Password aging is in effect for a user if the encrypted password is followed by a comma and a non-null string of characters from the above alphabet. (Such a string must be introduced in the first instance by the superuser.)

The first character of the age, *M* say, denotes the maximum number of weeks for that a password is valid. A user who attempts to log in after the password has expired is forced to supply a new one. The next character, *m* say, denotes the minimum period in weeks that must expire before the password can be changed. The remaining characters define the week (counted from the beginning of 1970) when the password was last changed. (A null string is equivalent to zero.) *M* and *m* have numerical values in the range 0–63 that correspond to the 64-character alphabet shown above (for example, / = 1 week;

z = 63 weeks). If $m = M = 0$ (derived from the string . or ..), the user must change the password at the next login (and the "age" disappears from the password file entry). If $m > M$ (signified by the string ./), only the superuser can change the password.

FILES

/etc/passwd
/etc/shadow
/etc/opasswd
/etc/oshadow

SEE ALSO

login(1), passwd(1), passmgmt(1M), a64l(3C), getpwent(3C), getspent(3X), group(4).

NAME

timezone - set default system time zone

SYNOPSIS

/etc/TIMEZONE

DESCRIPTION

This file sets and exports the time zone environmental variable **TZ**.

This file is ''dotted'' into other files that must know the time zone.

The syntax of **TZ** can be described as follows:

TZ → *zone*
 / *zone signed_time*
 / *zone signed_time zone*
 / *zone signed_time zone dst*
zone → *letter letter letter*
signed_time → *sign time*
 / *time*
time → *hour*
 / *hour : minute*
 / *hour : minute :*
dst → *signed_time*
 / *signed_time ; dst_date ,*
 / *; dst_date , dst_date*
dst_date → *julian*
 / *julian / time*

letter	→	*a / A / b / B / ... / z / Z*
hour	→	*00 / 01 / ... / 23*
minute	→	*00 / 01 / ... / 59*
second	→	*00 / 01 / ... / 59*
julian	→	*001 / 002 / .../ 366*
sign	→	*− / +*

EXAMPLES

The contents of **/etc/TIMEZONE** corresponding to the simple example below could be

```
#       Time Zone
TZ=EST5EDT
export TZ
```

A simple setting for New Jersey could be

```
TZ=EST5EDT
```

where **EST** is the abbreviation for the main time zone, **5** is the difference, in hours, between GMT (Greenwich Mean Time) and the main time zone, and **EDT** is the abbreviation for the alternate time zone. The most complex representation of the same setting, for the year 1986, is

TZ="EST5:00:00EDT4:00:00;117/2:00:00,299/2:00:00"

where **EST** is the abbreviation for the main time zone, **5:00:00** is the difference, in hours, minutes, and seconds between GMT and the main time zone, **EDT** is the abbreviation for the alternate time zone, **4:00:00** is the difference, in hours, minutes, and seconds between GMT and the alternate time zone, **117** is the number of the day of the year (Julian day) when the alternate time zone will take effect, **2:00:00** is the number of hours, minutes, and seconds past midnight when the alternate time zone will take effect, **299** is the number of the day of the year when the alternate time zone will end, and **2:00:00** is the number of hours, minutes, and seconds past midnight when the alternate time zone will end.

A southern hemisphere setting such as the Cook Islands could be

TZ="KDT9:30KST10:00;64/5:00,303/20:00"

This setting means that **KDT** is the abbreviation for the main time zone, **KST** is the abbreviation for the alternate time zone, KST is **9** hours and **30** minutes later than GMT, KDT is **10** hours later than GMT, the starting date of KDT is the **64**th day at **5** AM, and the ending date of KDT is the **303**rd day at **8** PM.

Starting and ending times are relative to the alternate time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year will be used and the time will be 2 AM. If the start and end dates are provided but the time is not provided, the time will be midnight.

NOTES

When the longer format is used, the **TZ** variable must be surrounded by double quotation marks as shown.

The system administrator must change the Julian start and end days annually if the longer form of the **TZ** variable is used.

Setting the time during the interval of change from the main time zone to the alternate time zone or vice versa can produce unpredictable results.

SEE ALSO

init(1M), ctime(3C), environ(5), rc2(1M).

NAME

disk - general disk driver

SYNOPSIS

#include <sys/types.h>
#include <sys/gdisk.h>
#include <sys/gdioctl.h>

DESCRIPTION

The CTIX special files **/dev/rdsk/c0d0s0** through **/dev/rdsk/c*x*d*x*s*x*** and **/dev/dsk/c0d0s0** through **/dev/dsk/c*x*d*x*s*x*** refer to CTIX device names and slices, where **c***x* is the controller number, **d***x* is the drive number, **s***x* is the slice number, and *x* is a hexadecimal digit. An **r** in the name indicates the character (raw) interface.

A disk is formatted with 512-byte physical sectors. I/O to a raw disk (**/dev/rdsk/c*x*d*x*s*x***) must be in a multiple of 512 bytes, or else an error (EINVAL) is returned. Logical block zero contains the *Volume Home Block* (VHB), which describes the disk. The VHB is structured to use two physical sectors as one logical block (1024 bytes).

The following structure defines the VHB:

```
struct vhbd {
    uint    magic;          /* S/MT disk format code */
    int     chksum;         /* adjustment so 32-bit sum starting
                               from magic for 1K bytes sums to -1 */
    struct gdswprt dsk;     /* specific description of this disk */
    struct partit partab[MAXSLICE]; /* partition table */
    struct resdes {         /* reserved area special files */
        daddr_t blkstart;  /* start logical block # */
        ushort nblocks;  /* length in logical blocks
                                (zero implies not present) */
    } resmap[8];
    /*      resmap consists of the following entries:
    *                       loader area
    *                       bad block table
    *                       dump area
    *                       down load image file
    *                       Bootable program,
    *                       size determined by a.out format. nblocks=1.
    */
    char    fpulled;        /* dismounted last time? */
    long    time;           /* time last came on line */
```

```
    struct gdswprt2 dsk2;  /* Drive specific parameters */
    char   minires[38];    /* for future mini/miti frame enhancements */
    char   sysres[292];    /* custom system area */
    struct mntnam mntname[MAXSLICE];
                           /* names for auto mounting; null
                            * string means no auto mount
                            * not used in mitiframe */
    char   userres[256];   /* user area */
};
struct gdswprt {
    char     name[6];  /* printf name */
    ushort   cyls;     /* the number of cylinders for this disk */
    ushort   heads;    /* number of heads per cylinder */
    ushort   psectrk;  /* number of physical sectors per track */
    ushort   pseccyl;  /* number of physical sectors */
                       /* per cylinder */
    char     flags;    /* floppy density and high tech drive flags */
    char     step;     /* stepper motor rate to controller - ST506 only */
    ushort   sectorsz; /* size of physical sectors (in bytes) */
};
struct gdswprt2 {
    short    wpccyl;   /* value to program for RWC/WPC - ST506 only */
    ushort   enetaddr[3];/* Ethernet station address -
                          * MiniFrame only */
    unchar   gap1;     /* Gap size on SMD drives */
    unchar   gap2;
    char     filler[28];
};
#define  sparesec    gap1    /* spare sectors per track  */
#define  sparecyl    gap2    /* spare tracks per cylinder */
#define  scinterleave wpccyl  /* interleave factor  */

struct partit{
    union {
        uint strk;/* start track number (new style) */
        struct {
            ushort strk;/* start track # */
            ushort nsecs;/* # logical blocks available to user */
        } old;
    } sz;
};
```

If a VHB is valid, *magic* is equal to VHBMAGIC and the 32-bit sum of the VHB's bytes is 0xFFFFFFFF (-1); *chksum* is the adjustment that makes the sum come out right.

The *dsk* structure describes the peculiarities of the disk, including deliberate deviations from the system standard. The *dsk.flags* field is the bitwise OR of zero or more of the following constants:

HITECH (ST506 only) If on, head select bit 3 is valid; if off, reduced write current is valid.

NEWPARTTAB If off, the old style slice (partition) table is in use; if on, the new style slice table is in use.

RWCPWC (ST506 only) If on, sets reduced write current/write precompensation.

HITECH Selects write precompensation.

FORMATEXTRA If on, the SMD drive is formatted with an extra sector on each track. (This sector is ignored by CTIX but is required for some disk drives, notably the Eagle-XP.)

The *dsk.step* field specifies a stepper motor rate for the ST506; uses 14 in this field.

The *partab* structure divides the disk into slices (partitions).

The *fpulled* field indicates whether an exchangeable disk was properly removed from the drive. The system sets this field to 1 when the disk is inserted in the drive. To clear *fpulled*, run *dismount*(1M).

The *mntname*, *minires*, and *userres* arrays are reserved for future use.

The *resmap* array describes the files that share slice 0 with the VHB. Provision is made for eight such files, but only five have assigned slots in *resmap*. Each *resmap* entry gives the starting location (logical block number) and length (logical blocks). A length of zero indicates that the file is not provided. The first five entries in *resmap* describe the following:

1. The loader. When the system is reset or turned on, the boot PROM loads the loader into the loader address and jumps execution to it. The function of the loader is to search for and load a program that will boot the system.

   On the S/640 and S/480, the loader searches the onboard tape, onboard (ST506) disks 0, 1, and 2, the VME, and the SCSI disks, in that order.

   On each disk, the loader checks for a CTIX kernel, which must be a CTIX executable object file called **/unix** in the file system in slice 1.

When the loader locates an appropriate program, it preserves the crash dump table, loads the program it found at the address it was linked at (0x0 if unknown), and executes it. If no disk contains an appropriate file, the loader continues searching until an appropriate disk is inserted.

2. The bad block table, which always begins at logical block 1 of the disk. Each logical block in the bad block table consists of a four-byte checksum followed by 127 bad block cells. The checksum is a value that makes the 32-bit sum of the logical block as 0xFFFFFFFF (-1). A bad block cell is defined by the following structure.

```
struct bbcell {
    ushort cyl;      /* the cylinder of the bad block */
    ushort badblk;   /* the physical sector address of the
                        bad block within the cylinder cyl */
    ushort altblk;   /* track number of alternate */
    ushort nxtind;   /* index into the cell array for next
                        bad block cell for this cylinder */
};
```

A single sequence of numbers, starting from zero, identifies the checksums and cells. For non-SCSI disks, in each cell in use, *cyl* identifies a cylinder that contains the bad block; *badblk* is the physical block offset within the cylinder of the bad block; *altblk* identifies the track that contains the alternate block; *nextind* (not used in S/MT) identifies the next cell for a bad block on the same cylinder or, if this is the last bad block, as zero.

SCSI disks perform their own bad block housekeeping. The bad block table contains only blocks that CTIX cannot read. At the next attempt to write to the bad block, CTIX issues a reassign block command to the SCSI drive. The drive then performs the bad block mapping for that sector, and the sector number is removed from the bad block table.

3. The dump area. After a reset or system crash, the boot PROM dumps processor registers, the memory map, a crash dump block, and the contents of physical memory, until it runs out of room in the dump area.

4. The download image area. The download images are described by a table at the beginning of the area. The area is described by the following array:

```
struct dldent {
        short d_strt; /* block displacement from download index */
        short d_sz; /* # of blocks for this entry */
};
```

The image number is the index for *dldent*. The *d_strt* field is the offset in bytes of the image from the beginning of the download image area; *d_sz* is the size in bytes of the image.

Slice 0 is called the Reserved Area. Only the Volume Home Block and the files described by *resmap* can be in the Reserved Area. A formatted disk used by a working system certainly has at least one more slice.

The *ioctl* system calls use the following structure:

```
struct gdioctl {
   ushort   status;              /* status */
   struct   gdswprt params;      /* description of the disk */
   struct   gdswprt2 params2;    /* more description of the disk */
   short    ctrltyp;             /* the type of disk controller */
   short    driveno;
};
```

where *status* is the bitwise OR of the following constants:

VALID_VHB	A valid Volume Header Block has been read.
DRV_READY	The disk is online.
PULLED	Last removal of disk from drive was not preceded by proper dismount.

params is a *gdswprt* structure, the same type used in the volume header block.

dsktype is equal to one of the following:

GD_WD1010	for Western Digital 1010 ST506 Controller
GD_WD2010	for Western Digital 2010 ST506 Controller
GD_RAMDISK	for RAM Disk Emulator
GD_SMD3200	for Interphase SMD3200 disk controller
GD_SCSI	for SCSI disk controller

CTIX understands the following disk *ioctl* calls:

ioctl(fd, GDIOCTYPE, 0)
: Returns **GDIOC** if *fd* is a file descriptor for a disk special file.

ioctl(fd, GDGETA, gdctl_ptr)
: *gdctl_ptr* points to a *gdioctl* structure; *ioctl* fills the structure with information about the disk.

ioctl(fd, GDSETA, gdctl_ptr)
: *gdctl_ptr points to a gdioctl* structure; *ioctl* passes the description of the disk to the disk driver. This is primarily meant for reading disks created by other kinds of computers.

ioctl(fd, GDFORMAT, ptr)
: *ptr* points to formatting information. The disk driver formats a track.

ioctl(fd, GDDISMNT)
: *ioctl* informs the driver that the user intends to remove the disk from the drive. When this system call successfully returns, the driver has flushed all data in the buffer cache and waited for all queued transfers to complete. The last transfer is to write out the Volume Home Block with the *fpulled* flag cleared. Once this call returns, the drive is inaccessible until a new disk is inserted.

ioctl(fd, GDPASSTHRU, arg)
: *arg* points to a disk driver-specific command block; *gd* passes the command to the specific disk driver untouched, and the disk driver performs the specific command.

SEE ALSO

iv(1), mknod(1M), ioctl(2).
S/Series CTIX Administrator's Guide.