-

NAME

20boot – install new 11/20 system

SYNOPSIS

20boot

DESCRIPTION

This shell command file copies the current version of the 11/20 program used to run the VT01 display onto the /dev/vt0 file. The 11/20 should have been started at its ROM location 773000.

FILES

/dev/vt0, /usr/mdec/20.o (11/20 program)

SEE ALSO

vt (IV)

NAME

boot procedures – UNIX startup

DESCRIPTION

The advent of the new system has changed the boot procedures. *These procedures apply only to C-language systems.*

How to start UNIX. UNIX is started by placing it in core starting at location zero and transferring to zero. There are various ways to do this. If UNIX is still intact after it has been running, the most obvious method is simply to transfer to zero.

The *tp* command places a bootstrap program on the otherwise unused block zero of the tape. The DECtape version of this program is called *tboot*, the magtape version *mboot.* If *tboot* or *mboot* is read into location zero and executed there, it will type '=' on the console, read in a *tp* entry name, load that entry into core, and transfer to zero. Thus the next easiest way to run UNIX is to maintain the UNIX code on a tape using *tp.* Then when a boot is required, execute (somehow) a program which reads in and jumps to the first block of the tape. In response to the '=' prompt, type the entry name of the system on the tape (we use plain 'unix'). It is strongly recommended that a current version of the system be maintained in this way, even if the first or third methods of booting the system are usually used.

The standard DEC ROM which loads DECtape is sufficient to read in *tboot*, but the magtape ROM loads block one, not zero. If no suitable ROM is available, magtape and DECtape programs are presented below which may be manually placed in core and executed.

A third method of rebooting the system involves the otherwise unused block zero of each UNIX file system. The single-block program *uboot* will read a UNIX pathname from the console, find the corresponding file on a device, load that file into core location zero, and transfer to it. The current version of this boot program reads a single character (either **p** or **k** for RP or RK, both drive 0) to specify which device is to be searched. *Uboot* operates under very severe space constraints. It supplies no prompts, except that it echos a carriage return and line feed after the **p** or **k.** No diagnostic is provided if the indicated file cannot be found, nor is there any means of correcting typographical errors in the file name except to start the program over. *Uboot* can reside on any of the standard file systems or may be loaded from a *tp* tape as described above.

The standard DEC disk ROMs will load and execute *uboot* from block zero.

The switches. The console switches play an important role in the use and especially the booting of UNIX. During operation, the console switches are examined 60 times per second, and the contents of the address specified by the switches are displayed in the display register. (This is not true on the 11/40 since there is no display register on that machine.) If the switch address is even, the address is interpreted in kernel (system) space; if odd, the rounded-down address is interpreted in the current user space.

If any diagnostics are produced by the system, they are printed on the console only if the switches are non-zero. Thus it is wise to have a non-zero value in the switches at all times.

During the startup of the system, the *init* program (VIII) reads the switches and will come up single-user if the switches are set to 173030.

It is unwise to have a non-existent address in the switches. This causes a bus error in the system (displayed as 177777) at the rate of 60 times per second. If there is a transfer of more than 16ms duration on a device with a data rate faster than the bus error timeout (approx 10μs) then a permanent disk non-existent-memory error will occur.

ROM programs. Here are some programs which are suitable for installing in read-only memories, or for manual keying into core if no ROM is present. Each program is position-independent but should be placed well above location 0 so it will not be overwritten. Each reads a block from the beginning of a device into core location zero. The octal words constituting the program are listed on the left.

DECtape (drive 0) from endzone:

```
012700          mov     $tcba,r0
177346
010040          mov     r0,-(r0)        / use tc addr for wc
012710          mov     $3,(r0)         / read bn forward
000003
105710  1:      tstb    (r0)            / wait for ready
002376          bge     1b
112710          movb    $5,(r0)         / read (forward)
000005
000777          br      .               / loop; now halt and start at 0
```

DECtape (drive 0) with search:

```
012700  1:      mov     $tcba,r0
177346
010040          mov     r0,-(r0)        / use tc addr for wc
012740          mov     $4003,-(r0)     / read bn reverse
004003
005710  2:      tst     (r0)
002376          bge     2b              / wait for error
005760          tst     -2(r0)          / loop if not end zone
177776
002365          bge     1b
012710          mov     $3,(r0)         / read bn forward
000003
105710  2:      tstb    (r0)            / wait for ready
002376          bge     2b
112710          movb    $5,(r0)         / read (forward)
000005
105710  2:      tstb    (r0)            / wait for ready
002376          bge     2b
005007          clr     pc              / transfer to zero
```

Caution: both of these DECtape programs will (literally) blow a fuse if 2 drives are dialed to zero.

Magtape from load point:

```
012700          mov     $mtcma,r0
172526
010040          mov     r0,-(r0)        / usr mt addr for wc
012740          mov     $60003,-(r0)    / read 9-track
060003
000777          br      .               / loop; now halt and start at 0
```

RK (drive 0):

```
012700          mov     $rkmr,r0
177414
005040          clr     -(r0)
005040          clr     -(r0)
010040          mov     r0,-(r0)
012740          mov     $5,-(r0)
000005
105710  1:      tstb    (r0)
002376          bge     1b
005007          clr     pc
```

RP (drive 0)

```
012700          mov     $rpmr,r0
176726
005040          clr     -(r0)
```

```
005040          clr     -(r0)
005040          clr     -(r0)
010040          mov     r0,-(r0)
012740          mov     $5,-(r0)
000005
105710  1:      tstb    (r0)
002376          bge     1b
005007          clr     pc
```

FILES

/usr/sys/unix – UNIX code
/usr/mdec/mboot – *tp* magtape bootstrap
/usr/mdec/tboot – *tp* DECtape bootstrap
/usr/mdec/uboot – file system bootstrap

SEE ALSO

tp(I), init(VII)

NAME

check – file system consistency check

SYNOPSIS

check [–**lsib** [numbers]] [filesystem]

DESCRIPTION

Check examines a file system, builds a bit map of used blocks, and compares this bit map against the free list maintained on the file system. It also reads directories and compares the link-count in each i-node with the number of directory entries by which it is referenced. If the file system is not specified, a check of a default file system is performed. The normal output of *check* includes a report of

The number of blocks missing; i.e. not in any file nor in the free list,
The number of special files,
The total number of files,
The number of large files,
The number of directories,
The number of indirect blocks,
The number of blocks used in files,
The highest-numbered block appearing in a file,
The number of free blocks.

The –**l** flag causes *check* to produce as part of its output report a list of the all the path names of files on the file system. The list is in i-number order; the first name for each file gives the i-number while subsequent names (i.e. links) have the i-number suppressed. The entries ‘‘**.**’’ and ‘‘**..**’’ for directories are also suppressed.

The –**s** flag causes *check* to ignore the actual free list and reconstruct a new one by rewriting the super-block of the file system. The file system should be dismounted while this is done; if this is not possible (for example if the root file system has to be salvaged) care should be taken that the system is quiescent and that it is rebooted immediately afterwards so that the old, bad in-core copy of the super-block will not continue to be used. Notice also that the words in the super-block which indicate the size of the free list and of the i-list are believed. If the super-block has been curdled these words will have to be patched. The –**s** flag causes the normal output reports to be suppressed.

The occurrence of **i** *n* times in a flag argument –**ii...i** causes *check* to store away the next *n* arguments which are taken to be i-numbers. When any of these i-numbers is encountered in a directory a diagnostic is produced, as described below, which indicates among other things the entry name.

Likewise, *n* appearances of **b** in a flag like –**bb...b** cause the next *n* arguments to be taken as block numbers which are remembered; whenever any of the named blocks turns up in a file, a diagnostic is produced.

FILES

Currently, /dev/rp0 is the default file system.

SEE ALSO

fs (V)

DIAGNOSTICS

There are some self-evident diagnostics like ‘‘can’t open ...’’, ‘‘can’t write ....’’ If a read error is encountered, the block number of the bad block is printed and *check* exits. ‘‘Bad freeblock’’ means that a block number outside the available space was encountered in the free list. ‘‘*n* dups in free’’ means that *n* blocks were found in the free list which duplicate blocks either in some file or in the earlier part of the free list.

An important class of diagnostics is produced by a routine which is called for each block which is encountered in an i-node corresponding to an ordinary file or directory. These have the form

b# complaint **; i=** *i#* **(***class* **)**

Here *b#* is the block number being considered; *complaint* is the diagnostic itself. It may be

blk if the block number was mentioned as an argument after **–b;**
bad if the block number has a value not inside the allocatable space on the device, as indicated by the devices's super-block;
dup if the block number has already been seen in a file;
din if the block is a member of a directory, and if an entry is found therein whose i-number is outside the range of the i-list on the device, as indicated by the i-list size specified by the super-block. Unfortunately this diagnostic does not indicate the offending entry name, but since the i-number of the directory itself is given (see below) the problem can be tracked down.

The *i#* in the form above is the i-number in which the named block was found. The *class* is an indicator of what type of block was involved in the difficulty:

sdir indicates that the block is a data block in a small file;
ldir indicates that the block is a data block in a large file (the indirect block number is not available);
idir indicates that the block is an indirect block (pointing to data blocks) in a large file;
free indicates that the block was mentioned after **–b** and is free;
urk indicates a malfunction in *check.*

When an i-number specified after **–i** is encountered while reading a directory, a report in the form

**ino; i=** *d#* **(***class* **)** *name*

where *i#* is the requested i-number. *d#* is the i-number of the directory, *class* is the class of the directory block as discussed above (virtually always ''sdir'') and *name* is the entry name. This diagnostic gives enough information to find a full path name for an i-number without using the **-l** option: use **–b** *n* to find an entry name and the i-number of the directory containing the reference to *n,* then recursively use **–b** on the i-number of the directory to find its name.

Another important class of file system diseases indicated by *check* is files for which the number of directory entries does not agree with the link-count field of the i-node. The diagnostic is hard to interpret. It has the form

i# delta

Here *i#* is the i-number affected. *Delta* is an octal number accumulated in a byte, and thus can have the value 0 through 377(8). The easiest way (short of rewriting the routine) of explaining the significance of *delta* is to describe how it is computed.

If the associated i-node is allocated (that is, has the *allocated* bit on) add 100 to *delta.* If its link-count is non-zero, add another 100 plus the link-count. Each time a directory entry specifying the associated i-number is encountered, subtract 1 from *delta.* At the end, the i-number and *delta* are printed if *delta* is neither 0 nor 200. The first case indicates that the i-node was unallocated and no entries for it appear; the second that it was allocated and that the link-count and the number of directory entries agree.

Therefore (to explain the symptoms of the most common difficulties) *delta* = 377 (–1 in 8-bit, 2's complement octal) means that there is a directory entry for an unallocated i-node. This is somewhat serious and the entry should be be found and removed forthwith. *Delta* = 201 usually means that a normal, allocated i-node has no directory entry. This difficulty is much less serious. Whatever blocks there are in the file are unavailable, but no further damage will occur if nothing is done. A *clri* followed by a *check –s* will restore the lost space at leisure.

In general, values of *delta* equal to or somewhat above 0, 100, or 200 are relatively innocuous; just below these numbers there is danger of spreading infection.

BUGS

Unfortunately, *check –l* on file systems with more than 3000 or so files does not work because it runs out of core.

Since *check* is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

It believes even preposterous super-blocks and consequently can get core images.

NAME

clri – clear i-node

SYNOPSIS

clri i-number [filesystem]

DESCRIPTION

Clri writes zeros on the 32 bytes occupied by the i-node numbered *i-number*. If the *file system* argument is given, the i-node resides on the given device, otherwise on a default file system. The file system argument must be a special file name referring to a device containing a file system. After *clri,* any blocks in the affected file will show up as ‘‘missing’’ in a *check* of of the file system.

Read and write permission is required on the specified file system device. The i-node becomes allocatable.

The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to zap an i-node which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the i-node is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated i-node, so the whole cycle is likely to be repeated again and again.

BUGS

Whatever the default file system is, it is likely to be wrong. Specify the file system explicitly.

If the file is open, *clri* is likely to be ineffective.

NAME

df – disk free

SYNOPSIS

df [filesystem]

DESCRIPTION

Df prints out the number of free blocks available on a file system. If the file system is unspecified, the free space on all of the normally mounted file systems is printed.

FILES

/dev/rf?, /dev/rk?, /dev/rp?

SEE ALSO

check(VIII)

BUGS

NAME

dump – incremental file system dump

SYNOPSIS

dump [key [arguments] filesystem]

DESCRIPTION

Dump will make an incremental file system dump on magtape of all files changed after a certain date. The argument *key*, specifies the date and other options about the dump. *Key* consists of characters from the set **iu0hds.**

- **i** the dump date is taken from the file **/etc/ddate.**
- **u** the date just prior to this dump is written on **/etc/ddate** upon successful completion of this dump.
- **0** the dump date is taken as the epoch (beginning of time). Thus this option causes an entire file system dump to be taken.
- **h** the dump date is some number of hours before the current date. The number of hours is taken from the next argument in *arguments.*
- **d** the dump date is some number of days before the current date. The number of days is taken from the next argument in *arguments.*
- **s** the size of the dump tape is specified in feet. The number of feet is taken from the next argument in *arguments.* It is assumed that there are 9 standard UNIX records per foot. When the specified size is reached, the dump will wait for reels to be changed. The default size is 1700 feet.

If no arguments are given, the *key* is assumed to be **i** and the file system is assumed to be **/dev/rp1.**

Full dumps should be taken on quiet file systems as follows:

```
dump 0u /dev/rp1
check -l /dev/rp1
```

The *check* will come in handy in case it is necessary to resore indiviidual files from this dump. Incremental dumps should then be taken when desired by:

```
dump
```

When the incremental dumps get cumbersome, a new complete dump should be taken. In this way, a restore requires loading of the complete dump tape and only the latest incremental tape.

FILES

/dev/mt0magtape
/dev/rp1default file system
/etc/ddate

SEE ALSO

restor, check(VIII), dump(V)

BUGS

-

NAME

ino – get the i-number of a file

SYNOPSIS

ino file ...

DESCRIPTION

The i-number of each file argument is printed. An i-number of zero is printed if a bad argument is given.

BUGS

NAME

mkfs – construct a file system

SYNOPSIS

/etc/mkfs special proto

DESCRIPTION

Mkfs constructs a file system by writing on the special file *special* according to the directions found in the prototype file *proto*. The prototype file contains tokens separated by spaces or new lines. The first token is the name of a file to be copied onto block zero as the bootstrap program (see boot procedures(VIII)). The second token is a number specifying the size of the created file system. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the i-list size in blocks (remember there are 16 i-nodes per block). The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user-id, the group id, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters –**bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or – to specify set-user-id mode or not. The third is **g** or – for the set-group-id mode. The rest of the mode is a three digit octal number giving the owner, group, and foreigner read, write, execute permissions (see *chmod* (I)).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a pathname whence the contents and size are copied.

If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers.

If the file is a directory, *mkfs* makes the entries **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

If the prototype file cannot be opened and its name consists of a string of digits, *mkfs* builds a file system with a single empty directory on it. The size of the file system is the value of *proto* interpreted as a decimal number. The i-list size is the file system size divided by 50. (This corresponds to an average size of three blocks per file.) The boot program is left uninitialized.

A sample prototype specification follows:

```
/usr/mdec/uboot
4872 55
d—-777 3 1
usr     d—-777 3 1
        sh      ———755 3 1 /bin/sh
        ken     d—-755 6 1
                $
        b0      b—-644 3 1 0 0
        c0      c—-644 3 1 0 0
        $
$
```

SEE ALSO

file system(V), directory(V), boot procedures(VIII)

DIAGNOSTICS

There are various diagnostics for syntax errors, inconsistent values, and sizes too small.

BUGS

It is not possible to initialize a file larger than 64K bytes.
The size of the file system is restricted to 64K blocks.

-

There should be some way to specify links.

NAME

mknod – build special file

SYNOPSIS

/etc/mknod name [**c**] [**b**] major minor

DESCRIPTION

Mknod makes a directory entry and corresponding i-node for a special file. The first argument is the *name* of the entry. The second is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (e.g. unit, drive, or line number).

The assignment of major device numbers is specific to each system. For reference, here are the numbers for the MH 2C-644 machine. Do not believe them too much.

Block devices:

0 RF fixed-head disk
1 RK moving-head disk
2 TC DECtape
3 TM magtape
4 RP moving-head disk
5 Vermont Research moving-head disk

Character devices:

0 KL on-line console
1 DC communications lines
2 PC paper tape
3 DP synchronous interface
4 DN ACU interface
5 core memory
6 VT scope (via 11/20)
7 DA voice response unit
8 CT phototypesetter
9 VS voice synthesizer
10 TIU Spider interface

SEE ALSO

mknod (II)

BUGS

NAME

mount – mount file system

SYNOPSIS

/etc/mount special file

DESCRIPTION

Mount announces to the system that a removable file system is present on the device corresponding to special file *special* (which must refer to a disk or possibly DECtape). The *file* must exist already; it becomes the name of the root of the newly mounted file system.

SEE ALSO

umount (VIII)

BUGS

Mounting file systems full of garbage can crash the system.

NAME

reloc – relocate object files

SYNOPSIS

reloc file octal [–]

DESCRIPTION

Reloc modifies the named object program file so that it will operate correctly at a different core origin than the one for which it was assembled or loaded.

The new core origin is the old origin increased by the given *octal* number (or decreased if the number has a '–' sign).

If the object file was generated by *ld*, the **–r** and **–d** options must have been given to preserve the relocation information and define any common symbols in the file.

If the optional last argument is given, then any *setd* instruction at the start of the file will be replaced by a no-op.

The purpose of this command is to simplify the preparation of object programs for systems which have no relocation hardware. It is hard to imagine a situation in which it would be useful to attempt directly to execute a program treated by *reloc.*

SEE ALSO

as(I), ld(I), a.out(V)

BUGS

NAME

restor – incremental file system restore

SYNOPSIS

restor key [arguments]

DESCRIPTION

Restor is used to read magtapes dumped with the *dump* command. The *key* argument specifies what is to be done. *Key* is a character from the set **trxw.**

- **t** The date that the tape was made and the date that was specified in the *dump* command are printed. A list of all of the i-numbers on the tape are also given.
- **r** The tape is read and loaded into the file system specified in *arguments.* This should not be done lightly (see below).
- **x** Each file on the tape is individually extracted into a file whose name is the file's i-number. If there are *arguments*, they are interpreted as i-numbers and only they are extracted.
- **w** In conjunction with the **x** option, before each file is extracted, its i-number is typed out. To extract this file, you must respond with **y.**

The **r** option should only be used to restore a complete dump tape onto a clear file system or to restore an incremental dump tape onto this. Thus

```
/etc/mkfs /dev/rp0 40600
restor r /dev/rp0
```

is a typical sequence to restore a complete dump. Another *restor* can be done to get an incremental dump in on top of this.

A *dump* followed by a *mkfs* and a *restor* is used to change the size of a file system.

FILES

/dev/mt0

SEE ALSO

dump, mkfs, check, clri (VIII)

DIAGNOSTICS

There are various diagnostics involved with reading the tape and writing the disk. There are also diagnostics if the i-list or the free list of the file system is not large enough to hold the dump.

BUGS

There is redundant information on the tape that could be used in case of tape reading problems. Unfortunately, *restor's* approach is to exit if anything is wrong.

Files that have been deleted are not removed when incremental tapes are loaded. It will be necessary to *check* the restored file system and *clri* any files that show up with a 201 delta diagnostic.

The current version of *restor* does not free space occupied by files that are overwritten. Thus a *check* will have to be performed to reclain the missing space.

NAME

su – become privileged user

SYNOPSIS

su

DESCRIPTION

Su allows one to become the super-user, who has all sorts of marvelous (and correspondingly dangerous) powers. In order for *su* to do its magic, the user must supply a password. If the password is correct, *su* will execute the Shell with the UID set to that of the super-user. To restore normal UID privileges, type an end-of-file to the super-user Shell.

The password demanded is that of the entry ‘‘root’’ in the system’s password file.

To remind the super-user of his responsibilities, the Shell substitutes ‘#’ for its usual prompt ‘%’.

SEE ALSO

sh (I)

NAME

sync – update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to insure file system integrity. See sync(II) for details.

SEE ALSO

sync(II)

BUGS

NAME

umount – dismount file system

SYNOPSIS

/etc/umount special

DESCRIPTION

Umount announces to the system that the removable file system previously mounted on special file *special* is to be removed.

SEE ALSO

mount (VIII)

DIAGNOSTICS

It complains if the special file is not mounted or if it is busy. The file system is busy if there is an open file on it or if someone has his current directory there.

BUGS

NAME

update – periodically update the super block

SYNOPSIS

update

DESCRIPTION

Update is a program that executes the *sync* primitive every 30 seconds. This insures that the file system is fairly up to date in case of a crash. This command should not be executed directly, but should be executed out of the initialization shell command file. See sync(II) for details.

SEE ALSO

sync(II), init(VII)

BUGS

There is a system bug which, it is suspected, may be aggravated by this program. Until further notice, *update* should not be run.