NAME

check − file system consistency check

SYNOPSIS

/etc/check [**−ib** [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 the default file systems 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 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

/etc/checklist

SEE ALSO

checklist(V), fs(V), clri(VIII), clrm(VIII), crash(VIII), fsdb(VIII), restor(VIII)
PWB/UNIX Operations Manual by M. E. Pearlman
Repairing Damaged PWB/UNIX File Systems by P. D. Wandzilak

DIAGNOSTICS

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 device’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 is given in the form

i# **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. The**−i** *n* option is used to find an entry name and the i-number of the directory containing the reference to *n,* then recursively use **−i** 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 *icheck −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

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.