Repairing Damaged PWB/UNIX File Systems

P. D. Wandzilak

March 1978

Bell Telephone Laboratories, Incorporated

Repairing Damaged PWB/UNIX File Systems

CONTENTS

Repairing Damaged PWB/UNIX File Systems

P. D. Wandzilak

Bell Laboratories
Piscataway, New Jersey 08854

1. INTRODUCTION

This document is intended to help PWB/UNIX* operations personnel repair file system damage. It assumes the reader has an understanding of [1,2]. A good understanding of the PWB/UNIX file system structure is also a prerequisite; in particular, it is assumed that the reader is familiar with the concept of *isize*, *fsize*, *ilist*, *inode*, *i-number*, and *small* and *large* files.

The task of repairing damaged file systems is not an easy or straightforward process. A considerable amount of thought and analysis of the problem must first be applied before any surgery is performed. Furthermore, situations do occur when a definite diagnosis of the problem is impossible; in such a situation, only experience and profound courage count.

This document is meant to be introductory in nature, and will not cover cases that exhibit a high degree of complexity or that require extensive knowledge and experience.

2. DEVICES—SPECIAL FILES

The PWB/UNIX Operating System distinguishes between two classes of devices: character devices and block devices. *Character devices* are basically byte oriented, e.g., terminals. *Block devices* are oriented to transferring larger groups of data, e.g., disks, magnetic tapes, etc. Each I/O device supported by PWB/UNIX is assigned a device class and major and minor device numbers. The major number is associated with the software driver for that device. The minor number specifies both the physical drive unit and a designated logical portion of the device. A *special file* is a PWB/UNIX file that has its major and minor device numbers recorded in addr[0] of the inode's control structure. Examples:

Character Devices:	*Block Devices:*
/dev/tty8	/dev/rp54
/dev/rrp24	/dev/rp24
/dev/rrp0	/dev/rp0

In the above examples, the devices /dev/rp0 and /dev/rrp0 both refer to the same physical device, but they have different access protocols.

3. CHECK DIAGNOSTICS

The first step in repairing file system damage is to find the extent of the damage. This is accomplished by executing the *check* command. Before the *check* command is invoked, the PWB/UNIX system should be rebooted and contained within the single-user environment; *check*(VIII) examines a file system and reports all inconsistencies. The normal output of *check* consists of the file-system device name followed by eight lines of summary information:

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;
the number of the highest block used;
the number of free blocks.

The following is an example of the normal *check* output:

---

* UNIX is a Trademark of Bell Laboratories.

```
# check /dev/rp0
/dev/rp0:
spcl        158
files       261
large        90
direc        29
indir        92
used       3630
last       6684
free       2923
```

Any additional information not shown above represents diagnostic messages produced by *check*. Diagnostic messages follow the file-system device name and precede the summary information, as shown below:

```
# check /dev/rp0
/dev/rp0:
   diagnostic messages:
        :
        :
spcl        158
files       261
large        90
direc        29
indir        92
used       3630
last       6684
free       2923
```

Each diagnostic message is a single entity. The user must have a thorough understanding of these messages in order to diagnose the problem and take proper corrective action.

The problem with most of the diagnostic messages is that their explanation is implicit rather than explicit. The following discussion clarifies these messages and advises the reader about their significance and importance.

3.1 Free-List Diagnostics

These diagnostics are the most common of all the *check* diagnostic messages. They pertain to areas within a file system that are designated as *free*. Each free area is linked together to form a chain, called the *free list*. The free list is dynamic and therefore sensitive to abrupt or abnormal system stops. When such a stop does occur, the free list can become curdled; this happens when the head of the in-core *free list* is not written out to disk. The *check* command recognizes these conditions and reports them via various diagnostic messages. The message "bad freeblock" means that a block number in the free list resides outside the range of available space. The message "dups in free" indicates duplication of disk blocks; this occurs when the same block or set of blocks in the free list appears in other files or in an earlier part of the free list. The last message type in this category concerns "missing" block numbers. A block is considered missing when it resides neither in the *ilist* (which contains control blocks for all files in a file system) nor in the remaining portion of the file system that consists of directories, ordinary files, and the free list.

When the *check* command is invoked upon a file system, and if the file system's free list is not consistent, one or more of these three diagnostic messages will appear. The output of check for a file system containing these types of free list errors is shown below:

```
# check /dev/rp0
/dev/rp0:
bad freeblock
20 dups in free
2000 missing
spcl        158
files       261
large        90
direc        29
indir        92
used       3630
last       6684
free        923
```

3.2 Block Diagnostics

This section describes diagnostic messages that concern disk blocks of both ordinary files and directories. An *inode* is a 32-byte control block structure that identifies various attributes of each file or directory (see *File System*(V)). All inodes reside in the portion of the file system identified as the *ilist*. Each *inode* in the *ilist* is numbered and identified by an i-number. An *i-number* uniquely identifies a file in a file system. This type of message is of the form:

b# error; inode = i# (block type)

where *b#* indicates the actual *decimal* block number, *error* is the type of diagnostic in a mnemonic form, and *i#* represents the actual i-number associated with the particular block number.

These diagnostics are based upon information contained in the super-block of the file system; although limited precautions such as range checks, size checks on both *isize* and *fsize*, etc., are taken, erroneous diagnostics can occur due to wrong data values in the super-block. *Isize* (the contents of the first word in the super-block of the file system) is the block size of the *ilist* region. *Fsize* (the contents of the second word in the super-block of the file system) is the block size of the entire file system.

Descriptions for each of the *error* mnemonics are given below:

bad	The block number contains a value outside the allowable space on the file system. The block number cannot be less than (*isize*+ 2) or greater than *fsize*.
dup	The block number is assigned to a previously-encountered file. This message represents only this particular reference to this block. The procedure for finding all such references is described in the next section on file patching.
din	The block is a member of a directory entry and contains an i-number outside the range defined by the *ilist* size of the file system. The size of the *ilist* is calculated by multiplying the value of *isize* by the number of inodes per disk block. There are currently 16 inodes in a 512 byte disk block.
blk	The block number was used as an argument (*-b* option) in *check*.

The *block type* identifies the kind of block involved:

free	The block is not allocated.
data(small)	The block is a data block in a small file. A *small* file consists of eight or fewer disk blocks that directly contain the block addresses of data.
data(large)	The block is a data block in a large file. In a *large* file, each of the eight block addresses may reference an *indirect* block of up to 256 addresses of blocks that, in turn, constitute the file itself.
indirect	The block is an indirect block in a large file.

Examples of these types of diagnostics messages are:

```
# check -b 53 /dev/rp0
/dev/rp0:
53 blk; inode= 0(free)
53 bad; inode= 0(free)
6 dups in free
3400 missing
spcl        159
files       254
large        88
direc        18
indir        88
used       3067
last       6681
free         92

# check /dev/rp54
/dev/rp54:
36638 dup; inode= 2205(data (small))
14563 dup; inode= 2208(data (small))
58318 din; inode= 2208(data (small))
58318 din; inode= 2208(data (small))
spcl          0
files      4970
large       742
direc       332
indir       767
used      44412
last      64666
free      19658
```

3.3 Inode Diagnostics

The third class of messages that will be discussed involves individual files, for which the actual number of directory references does not agree with the value contained in their link-count field in the superblock of the file system. The format of these messages is as follows:

i# error-indicator

The i# represents the i-number of the inode. The error indicator consists of a byte whose octal value ranges from 0 to 377_8. The algorithm that *check* employs for computing the error indicator is explained below:

a. 100_8 is added to the error indicator if the particular inode is allocated. The inode is disregarded if it is not allocated and its *mode* (permissions of the file) is zero.

b. An additional 100_8 is added to the error indicator if the link-count field of the inode is non-zero; the value of the link-count field is also added to the error indicator.

c. For each directory entry which references that particular inode, 1 is *subtracted* from the error indicator.

Only inodes whose error-indicators are neither 0 (unallocated) nor 200 (consistent file) are reported and flagged as requiring additional attention. The most common occurrences contain error-indicators of 100, 177, 201, 377. Example:

```
# check /dev/rp54
/dev/rp54:
    15   100
    23   201
    39   177
   213   377
spcl          0
files      4970
large       742
direc       332
indir       767
used      44412
last      64666
free      19658
```

Because of their frequent occurrence, a further explanation of the four diagnostics in the above example are given below.

3.3.1 Type-100 Error. In the case where the error-indicator is 100, the inode usually is allocated, but its link count is zero. This can be made evident by retracing through the steps that the *check* algorithm employs for computing the error-indicator value and by the example below, which illustrates the structure of such an inode in raw data format. *File System*(V) describes the complete structure of an inode and should be consulted for a more complete explanation. Example (see *fsdb*(VIII)):

```
# fsdb /dev/rp0
15i
i#: 15 md:a-----f------- ln: 0 uid: 0 gid: 0 s0: 0 s1: 0
a0: 1562 a1: 0 a2: 0 a3: 0 a4: 0 a5: 0 a6: 0 a7: 0
at: Thu May 26 13:31:06 1977 mt: Thu May 26 13:31:06 1977
```

where *md* is the mode (indicating allocation) and *ln* represents the link-count field.

This particular error is not catastrophic to the file system and does not necessitate immediate attention. The data blocks are still allocated, e.g., 1 block at 1562. The file is not referenced by any directory; hence, it is nameless. This error normally occurs as a result of a pipe being active when the system crashes.

3.3.2 Type-201 Error. Error-indicator 201 implies that an i-number does not appear in enough directories (201—missing in one directory, 202—missing in two directories, etc.). Such an occurrence is similar to the previous case in that its nature is not serious and does not warrant immediate attention. The data blocks associated with the inode will remain intact, but will not be available until the problem is remedied. Example:

```
# fsdb /dev/rp0
23i
i#: 23 md: a----rw------ln: 1 uid: 11 gid:1 s0: 0 s1: 3050
a0: 660 a1: 671 a2: 724 a3: 824 a4: 34 a5: 1056 a6: 0 a7: 0
at: Thu May 12 16:05:10 1977 mt: Wed May 25 10:53:11 1977
```

The eight address blocks (*a0-a7*) contain the actual raw data for this particular example. A more detailed description of these eight address blocks can be found in *File System*(V).

3.3.3 Type-377 Error. Error-indicator 377 (namely, -1 represented in two's complement format) indicates that an unallocated inode appears as an entry in a directory. This situation is *serious* and should be taken care of immediately. The major implication in this case is that the inode is not allocated and the link count is zero. This means that the system might reassign the inode to a new file and spread the infection to other parts of the file system. Example:

```
# fsdb /dev/rp0
213i
i#: 213 md: 0 ln: 0 uid: 11 gid: 1 s0: 0 s1: 0
a0: 0 a1: 0 a2: 0 a3: 0 a4: 0 a5: 0 a6: 0 a7:0
at: Thu May 12 16:05:10 1977 mt: Wed May 25 10:53:11 1977
```

where *md* indicates no allocation and *ln* represents the link count.

3.3.4 Type-177 Error. Error-indicator 177 indicates that the inode appears as an entry in one more directory than indicated by its link count. This problem, like the above, is serious and immediate attention should be given to correcting it before the infection spreads to other parts of the file system. For instance, removing an inode that has a 177 error indicator could turn it into one with a 377 error indicator. Example:

```
# fsdb /dev/rp0
39i
i#: 39 md: ad----rwxr-xr-x ln: 1 uid: 11 gid: 1 s0: 0 s1: 160
a0: 753 a1: 0 a2: 0 a3: 0 a4: 0 a5: 0 a6: 0 a7: 0
at: Thu May 12 16:05:10 1977 mt: Fri June 25 10:53:11 1977
```

Where *ln* represents the link-count field.

4. REPAIRING FILE SYSTEM DAMAGE

This section provides guidance and some methods for resolving the problems that lead to the various diagnostic messages discussed in Section 3 above. The solutions presented here are examples of actual occurrences.

4.1 Free-List Diagnostics

The first class of diagnostic messages are Free List Diagnostics. These types of messages are the most common and the least complex. They basically involve the portion of a file system known as the super-block. The super-block contains the sizes of both the *isize* and *fsize* and system tables of up to 100 free blocks and free i-numbers. Also, a pointer to a linked list of blocks, the free list, is contained within the super-block. The "root" (/) file system's super-block is considered as a special case, because the root file system is always mounted whenever PWB/UNIX is running. When a file system is mounted as a file system, a copy of its super-block is loaded into memory. Normally, before any actual repair work can be performed, the system should be set to the single-user environment. This will ensure no outside interference, as well as prevent spreading the infection to other file systems, because the root file system is normally the only file system mounted in single-user mode.

All repair work on a file system is made in its place of residence (normally a disk pack). Therefore, upon completion of repair work on the free list of the *root* file system, an *immediate reboot* of the system is required *without* a *sync*(I). The reboot updates the in-memory super-block of the root file system with the newly-repaired copy. Omitting this step guarantees disastrous results.

4.1.1 Root File System. We show below a simple example involving the root file system and the necessary repair steps that are to be followed:

1.

```
# check /dev/rp0
/dev/rp0:
1 dups in free
1 missing
spcl        159
files       270
large        97
direc        18
indir        98
used       3443
last       6681
free       3109
```

The above represents the present condition of the "root" file system after the system was rebooted and is in single-user mode. The error messages (dups in free, etc.) can now be repaired by invoking the command *icheck*, which searches through the specified file system and reconstructs the free list. The syntax for *icheck* is as follows:

```
# icheck -s[drive type] special-file
```

where:

s	Indicates that the free list is to be built.
drive type	Where the drive type corresponds to the numbers 3 through 6 (*RP03, RP04, ... RP06*) and, depending upon which type is specified, the free blocks are then built in that order.
special file	File system device name (i.e., /dev/rp0, etc.).

The steps needed to repair the "root" file system in the above condition are as follows:

2.

```
# icheck -s4 /dev/rp0
/dev/rp0:
   :
   :
```

3.

```
# #00 = UNIX
system pwbb0215
Restricted rights:
Use, duplication ...

Available user memory = 6917 × 32 words
```

4.

```
# check /dev/rp0
/dev/rp0:
spcl        159
files       270
large        97
direc        18
indir        98
used       3443
last       6681
free       3109
```

In step 2 above, the command *icheck* is invoked to reconstruct the free list. Step 3 shows that the system is then *halted* and then *rebooted*, because the "root" file system was involved as explained above. The remaining step, a recheck of the file system, is performed in order to make sure that the repair succeeded. The count of free blocks between steps 1 and 4 remained the same because the error message "1 dups in free" offsets the succeeding error message "1 missing" in step 1.

This procedure should be followed for all possible combinations of error messages that pertain to the "free list". The only exception to this procedure concerns step 3 above, which involves a reboot of the system if the root file system is specified.

4.1.2 Non-root File System. Below is an example that shows this procedure used on a non-root file system:

1.

```
# check /dev/rp44
/dev/rp44:
43 missing
spcl          0
files      9207
large      1178
direc       673
indir      1185
used      58807
last      64999
free       5433
```

The above output from *check* represents the condition of the file system before any correctional steps are applied. The steps needed to repair a "non-root" file system in the above condition are as follows:

2.

```
# icheck -s4 /dev/rp44
/dev/rp44:
    :
    :
```

3.

```
# check /dev/rp44
/dev/rp44:
spcl          0
files      9207
large      1178
direc       673
indir      1185
used      58807
last      64999
free       5476
```

The 43 blocks that were shown as "missing" in step 1 are now part of the reconstructed free list.

4.2 Block Diagnostics

The next class of diagnostic messages is Block Diagnostics. Section 3 described the two distinct types that make up this category. Their respective formats are as follows:

1. b# error; inode = i# (block type)
2. i# error-indicator

This type requires checking each disk block in a file system that is associated with a particular inode.

4.2.1 "DUP" Diagnostic. A typical example is the error message *dup* followed by an i-number and the type of disk block. We now show a procedure used to resolve this kind of problem:

1.

```
# check /dev/rp2
/dev/rp2:
25511 dup; inode = 2201 data (small)
spcl          0
files      3987
large       858
direc       338
indir       883
used      39647
last      52660
free      12952
```

2. The initial check of the file system shows a *dup* error message; this message indicates that the allocated block number (25511) contains more than one *reference point.* This reference point can refer to either a previous file or possibly the same file. The first step is to find all the reference points (i-numbers of files), that are referencing this allocated disk block. This is done by:

```
# check -b 25511 /dev/rp2
```

This will give a complete listing of all the i-numbers that *reference* the disk block 25511.

```
/dev/rp2:
25511 blk; i=1105 data(small)
25511 blk; i=2201 data(small)
25511 dup; i=2201 data(small)
spcl          0
files      3987
large       858
direc       338
indir       883
used      39647
last      52660
free      12952
```

3. The above output shows two i-numbers, 1105 and 2201, that reference the disk block 25511. The names associated with these i-numbers can be determined by invoking the command *ncheck*(VIII). *Ncheck* lists the given i-numbers with their respective names:

```
# ncheck -i 1105 2201 /dev/rp2
/dev/rp2:
1105      /hasp/info/logx484
2201      /hasp/testjob
#
```

4. One now has to determine which of the two i-numbers is the rightful owner of the disk block 25511. Sometimes this is impossible; the only possible solution is to remove *all* the files named in step 3. In this particular example, uncertainty is not involved; the data contents of disk block 25511 are displayed and give clues in identifying the correct owner, as follows:

```
# fsdb /dev/rp2
25511b,64c
61647000: / / t e s t j o b   j o b     ( 6
61647020: 2 7 0 , r 8 4 0 ) , r j e ,   c l
61647040: a s s = x\n / / *         u s r
61647060: = ( h a s p , j o b )\n / / i e
```

The first line above invokes *fsdb*(VIII) with the names of the file system as its argument. *Fsdb* provides an efficient method for scanning an inode or a disk block. The second line is supplied by the user as input to *fsdb;* the "25511" is the (decimal) number of the block that will be inspected. The"b" expands the block number to a full block address, (i.e., 61647000). The "64" indicates that 64 characters are to be printed. The letter "c" causes the 64 characters to be printed in *character mode.* The output resembles the contents of the file /usr/hasp/testjob.

5. Once the correct owner has been established, the file can then be removed, but only after the file system /dev/rp2 is *mounted.* Steps 1 through 4 were accomplished while the file system /dev/rp2 was *unmounted* or *inactive:*

```
# mount /dev/rp2 /mnt
WARNING!! - mounting: <c2> as </mnt>
# rm -f /mnt/hasp/info/logx484
```

At this point, the file system "/c2" can then be *unmounted* (first line below), its free list reconstructed (next two lines), and checked again:

```
# umount /dev/rp2
# icheck -s4 /dev/rp2
/dev/rp2:
    :
    :
# check /dev/rp2
/dev/rp2:
spcl          0
files      3986
large       858
direc       338
indir       883
used      39646
last      52660
free      12952
```

The process of identifying the principal owner (i-number) of the disk block sometimes fails. When this occurs, one possible solution is to remove *all* files that reference that disk block (25511). The procedure in this case remains the same as before, with the exception that step 4 and the last line of step 5 are replaced with the following:

```
# rm -f /mnt/hasp/info/logx484
# rm -f /mnt/hasp/testjob
```

All files that are removed should be restored from the latest backup medium for that file system. The owner of each such file must then be notified of the problem and be given the full path name of each restored file.

An alternative procedure would be to first recopy all the files involved to temporary files, and then remove the original files. The owners should be notified that their files have been renamed and that they possibly contain contaminated information.

4.2.2 An Example of a More Complex "DUP" Diagnostic. The above case represents a case that does not involve massive damage. However, when massive damage is present, the repair process is substantially more complex, as shown in the following example:

```
# check /dev/rp34
/dev/rp34:
16705 dup; inode = 12960 (data (large))
8270 dup; inode = 12960 (data (large))
8224 dup; inode = 12960 (data (large))
13112 dup; inode = 12960 (data (large))
13873 dup; inode = 12960 (data (large))
13360 dup; inode = 12960 (data (large))
8224 dup; inode = 12960 (data (large))
12320 dup; inode = 12960 (data (large))
21332 dup; inode = 13052 (data (large))
8242 dup; inode = 13991 (data (small))
12342 dup; inode = 15175 (data (small))
18516 dup; inode = 15220 (data (small))
16967 dup; inode = 16164 (data (large))
21326 dup; inode = 16948 (data (small))
19785 dup; inode = 17305 (data (large))
20037 dup; inode = 17305 (data (large))
1 dups in free
14 missing
spcl        0
files   15428
large     492
direc     701
indir     525
used    52193
last    64989
free    11485
```

The inode 12960 claims to be the owner of several disk blocks. This inode is the most likely candidate that should be removed in order to restore consistency to the file system. The following steps will repair the file system damage and restore the file system to a working condition:

```
# ncheck -i 12960 /dev/rp34
/dev/rp34:
12960 /9452/sita/pscsxs/aaaacsh1a/AAN---831604--
# mount /dev/rp34 /mnt
WARNING!! - mounting: <b4> as </mnt>
# rm -f /mnt/9452/sita/pscsxs/aaaacsh1a/AAN---831604--
# umount /dev/rp34
# icheck -s4 /dev/rp34
/dev/rp34:
    :
    :
# check /dev/rp34
/dev/rp34:
spcl        0
files   15427
large     491
direc     701
indir     524
used    51936
last    64989
free    11500
```

The output of the last *check* command above reveals that the damaged file system, /dev/rp34, was repaired and is now consistent. However, the procedure cannot guarantee that the data blocks associated with the other affected files contain valid information. The procedure can be expanded to include the necessary steps to identify the path names of all the affected files in the above list. The owners can

then be notified that their files may contain invalid data. The additional steps replace the *ncheck* command in the above procedure.

Each block number reported as a *dup* should be checked. This will then identify all the i-numbers referencing a duplicate block:

```
# check -bbb ...b 16705 8270 8224 ... 20037 /dev/rp34
```

Where *n* occurrences of b (-bbb...b) cause the next *n* arguments to be interpreted as block numbers. Each unique i-number reported in the previous step can then be converted into a path name.

```
# ncheck -i [i-numbers] /dev/rp34
```

4.2.3 "BAD" Diagnostic. The message *bad* implies that the block number contains a value outside the *allowable space* on the file system. *Allowable space* is defined as the number of disk blocks logically addressable, from *isize* + 2 to the limit specified by the super-block entry *fsize*.

These messages usually occur for a small file that was in the process of being transformed into a large file at the time of the crash. The mode field in the inode control structure indicates a large file, but the address block references a data block address instead of an indirect block address.

Because the bad block and the file are normally unrelated to one another, the contaminated file must be removed. The example below will be used as a model for developing a procedure for this kind of problem:

```
# check /dev/rp24
/dev/rp24:
49802 bad; inode = 1191 (data(large))
49810 bad; inode = 1191 (data(large))
128 bad; inode = 1191 (data(large))
32768 dup; inode = 1191 (data(large))
32768 dup; inode = 1191 (data(large))
40960 bad; inode = 1191 (data(large))
1 bad; inode = 1191 (data(large))
436 bad; inode = 1191 (data(large))
4 dups in free
73 missing
spcl        0
files    4378
large     854
direc     299
indir     859
used    31218
last    65475
free     4118
```

Once the initial check of the file system has been performed, as above, the following steps must be performed:

1. Unallocate the i-number 1191. This will unallocate the bad blocks and leave the i-number as an non-existent file with its directory entry and name field intact. This is done by using *clrm*(VIII), (rather than *rm*(I), because the system checks for bad block numbers when *rm* is used, and might not remove the file):

   ```
   # clrm /dev/rp24 1191
   ```

2. Check the file system to insure that the bad messages were reduced into a simpler diagnostic:

```
# check /dev/rp24
/dev/rp24:
4 dups in free
14 missing
   1191  377
spcl          0
files      4377
large       853
direc       299
indir       858
used      31215
last      35989
free       4118
```

3. Convert the i-number 1191 from step 2 into a path name:

```
# ncheck -i 1191 /dev/rp24
/dev/rp24:
1191    /adm/core
```

4. Mount the file system:

```
# mount /dev/rp24 /mnt
WARNING!! - mounting: <a1> as </mnt>
```

5. Remove the name of the non-existent file:

```
# rm -f /mnt/adm/core
```

6. Unmount the file system:

```
# umount /dev/rp24
```

7. Reconstruct the free list to eliminate the "dups in free" that result from step 5.

```
# icheck -s3 /dev/rp24
```

8. Check the file system again:

```
# check /dev/rp24
/dev/rp24:
spcl          0
files      4377
large       853
direc       299
indir       858
used      31215
last      35989
free       4129
```

9. The contaminated file that was removed in step 5 should be restored from the latest back-up medium for that file system and its owner notified.

4.2.4 "DIN" Diagnostic. The message *din* means that a block number is a member of a directory entry containing an i-number that resides outside the range of the *ilist* size of the file system; *isize* (the contents of the first word in the super-block of the file system) is the block size of the *ilist* region.

These messages normally occur when a directory contains an address block that references text data instead of directory entries. A *directory entry* consists of an i-number and a 14-character name. The example below will be used as a model for developing a procedure for this kind of problem.

```
# check /dev/rp24
/dev/rp24:
7902 din; inode = 15 (data(small))
1 dups in free
1 missing
       1  201
       2  201
       3  201
       5  201
       7  201
       9  201
      13  201
      14  201
      15  201
      33  201
     168  201
     169  201
     172  201
     340  201
     488  201
     808  201
    2975  201
    3976  201
    3977  201
    4099  201
    4335  201
    4364  201
    4547  201
    5002  201
    8202  377
    8224  342
spcl          0
files      4769
large       870
direc       251
indir       879
used      33545
last      35989
free       1798
```

The first step is to position *fsdb* at the beginning of the inode structure for i-number 15 in file system /dev/rp24.

.1.
```
# fsdb /dev/rp24
15i
i#: 15 md:ad----rwxr-xr-x ln: 3 uid: 2 gid: 0 s0: 0 s1: 560
a0: 7902 a1: 3314 a2: 0 a3: 0 a4: 0 a5: 0 a6: 0 a7: 0
at: Fri May 27 17:30:02 1977 mt: Fri May 27 07:01:02 1977
```

Where 15i is the offset into the file system. The first address field (a0) contains the same block address as was reported by the initial *check* above. The block 7902 and its data contents will be further investigated.

2.
```
7902b.p32d
 d0:  8224          Field.
 d1:  8224
 d2:  8224
 d3:  8224
```

```
 d4:  8224
 d5:  8224          IF
 d6:  8224
 d7:  8224
 d8:  8224
 d9:  8224
d10:  8224          Only
d11:  8224
d12:  8224
d13:  8224
d14:  8224
d15: 28448          NE/n
d16:  8224
d17:  8224
d18:  8224
d19:  8224          PM
d20:  8202
d21:  8224
d22:  8224
d23:  8224
d24:  8224          IS
d25:  8224
d26:  8224
d27:  8224
d28:  8224
d29:  8224          DESIRED
d30:  8224
d31:  8224
q
```

In the above, the "7902b.p32d" displays the contents of block 7902 as directory entries: q terminates this command. The block 7902 is obviously a data block and not a block of directory entires. The number 8224 in each instance is *interpreted* as an i-number and accounts for the diagnostic message 342 (last diagnostic in the output of the *check* command above). The *din* diagnostic occurred at the fifteenth entry (d15:, in step 2 above) because the number, 28448, exceeds the boundary limitations of the *ilist size* (the *ilist size* for file system, /dev/rp24, was defined as 655 blocks, with each block containing a maximum of 16 inodes for a maximum of 10480 i-numbers). This shows that the directory block is damaged. Because directories change infrequently, as a rule, there is, in all probability, a good copy on the most recent backup pack.

The next step is to physically mount (on another drive) the latest backup copy for the file system /dev/rp24 and then to replace the contaminated block address (7902) on the original file system with the corresponding inode's block address from the backup copy.

3. # fsdb (backup file system)

```
15i
i#: 15 md: ad----rwxr-xr-x ln: 3 uid: 2 gid: 0 s0: 0 s1: 560
a0: 1344 a1: 3314 a2: 0 a3: 0 a4: 0 a5: 0 a6: 0 a7: 0
at: Fri May 27 07: 01: 01 1977 mt: Fri May 27 07: 01: 01 1977
```

Note that we used the same offset (15i) into the backup file system as we did above for the damaged file system.

4. The contents of the first address block (1344) in Step 3 are displayed below:

```
1344b.p32d
d0:     15   .
d1:      1   ..
d2:     14   dtablezero
d3:     13   SA
d4:   5002   dtable
d5:    808   lock
d6:      0   dtbsv
d7:      9   dailyacct
d8:   4099   diskusage
d9:      7   wtmp
             :
             :
d31:     0   x
q
```

In the above, the "1334b.p32d" displays the contents of block 1344 as directory entries; q terminates this command. Note the difference between the above output and the output in step 2; also the same i-numbers that were reported as 201 type errors in step 2 will have valid directory entries once the block transformation is made.

The next step is to determine if the block number (1344) on the original file system is unallocated. This step is necessary in order to prevent duplicate block numbers in files. The following command line will perform this function:

```
# check -b 1344 /dev/rp24 (original file system)
```

If the response from the above command is

```
1344 blk; inode = 0 (free)
```

then block number 1344 can be used. Otherwise, the block is already allocated and an alternate block number must be used from the free list.

For the case where the same block number is to be reused, there is a high probability that the contents within the block have not been altered and thus eliminates the need for recopying the block before we use it. The block 1344 on the original file system should be inspected; its contents should resemble the data contents in Step 4. If differences between the contents of the two blocks are present, it is recommended that the block be re-copied with *bcopy*(VIII) before using the block. However, the contaminated block address (7902) on the original file system must be changed to the new block address (e.g.,1344):

5. # fsdb /dev/rp24 (original file system)
   15i.a0 = 1344
   q

The steps below will reconstruct the *free* list and re-check the file system for accuracy:

6. # icheck -s3 /dev/rp24

7. # check /dev/rp24
   /dev/rp24:
   spcl 90
   files 4769
   large 870
   direc 251
   indir 879
   used 33545
   last 35989
   free 1798

4.3 Inode Diagnostics

The last class of error diagnostics that will be discussed here has the following format:

i# error-indicator

Section 3 defined the framework for this category of diagnostics and identified the types that occur most frequently. All instances will now be examined separately and will illustrate methods that can be employed to repair them. Although each of these types of errors is different in definition and complexity, they do share some similarity with respect to their repair processes.

4.3.1 Type-100 Error. We first discuss the "type-100" errors ("type-201" errors are discussed in Section 4.3.4). The example below shows both 100 and 201 type errors.

```
# check /dev/rp0
/dev/rp0:
   426   100
   427   201
   429   100
   431   201
   435   100
   438   100
spcl        174
files       268
large        92
direc        29
used       2904
last       6681
free       3661
```

The above example pertains to the "root" file system. This fact is important, because the root file system is a special case: it requires an immediate reboot of the system after any alterations are made in its super-block.

The "type-100" error usually implies that the inode is allocated, but that its link-count is zero. The process for repairing "type-100" errors is to make the inode unallocated. The command *clrm*(VIII) (clear inode mode) performs this function. The procedure is as follows:

1. clrm file-system [i-numbers ...]

   e.g.:

   ```
   # clrm /dev/rp0 426 429 435 438
   ```

2. ```
   # icheck -s4 /dev/rp0
   /dev/rp0:
       :
       :
   # #00= unix
   system pwbd0127
   Restricted rights:
       :
       :
   Available user memory = 6910 × 32 words
   ```
   ```

```
3. # check /dev/rp0
   /dev/rp0:
      427   201
      431   201
   spcl        174
   files       264
   large        92
   direc        29
   used       2904
   last       6681
   free       3670
```

The first command above reconstructs the root free list; because the root file system is involved, the system is then rebooted in single-user mode, and re-checked to ensure that all "type-100" errors have been taken care of. This procedure can be applied to any *inactive* file system. The rebooting after the *icheck* is needed when the "root" file system is involved.

4.3.2 Type-377 Error. The next type in this class of diagnostic messages consists of "type-377" errors (-1 represented in two's complement form). This particular error type indicates that an unallocated inode is represented as an entry in a directory as shown in the example below:

```
# check /dev/rp14
/dev/rp14:
37 missing
   1992  377
spcl          0
files     11364
large      1069
direc       817
indir      1074
used      47872
last      64998
free      15864
```

Because this type of error is considered highly communicable in spreading the infection to other files, it is crucial that this be attended to immediately. This error can also cause diagnostics with error indicator values of 376, 375, etc. Instances where the value of the error indicator is other than 377 (i.e., 376, 375, etc.), are most likely to be *directories* or *multiply-linked files*. When these conditions arise, the repair procedures become more complex and demand a greater amount of knowledge and experience. These errors are also very serious and warrant immediate attention.

The repair process for a multiply-linked file is identical to the procedure for repairing an ordinary file that contains an error indicator value equal to 377_8. The differences are that several file names replace the single name referenced in both step 1 and step 3 of the procedure below and that the error indicator should equal values of 376, 375, etc.

When a directory is involved, the error condition becomes compounded, because the directory itself and its contents (*files* and *subdirectories*) are also affected. The first step is to determine the complete path name of the directory. Once the path name has been determined, the entire directory can then be removed and later restored from the latest available backup medium.

The process of converting an i-number into a path name is accomplished by the command *ncheck*. This command will terminate prematurely when a directory that is needed to complete the construction of the path name is non-existent. The path name must then be constructed manually, which is difficult and error-prone.

The procedure for repairing an ordinary file that contains an error indicator of 377 is as follows:

1. The first step is to convert the i-number (1992) into a character name:

```
# ncheck -i 1992 /dev/rp14
/dev/rp14:
1992  /fl/payload/r43117
```

2. Once the path name is determined, the file system must now be mounted as an active file system:

```
# mount /dev/rp14 /mnt
WARNING!! - mounting: <f3> as </mnt>
```

3. The name of the non-existent file can now be removed:

```
# rm -f /mnt/fl/payload/r43117
```

4. The file system is then unmounted:

```
# umount /dev/rp14
```

5. The file system is checked again for accuracy:

```
# check /dev/rp14
/dev/rp14:
spcl          0
files-    11364
large      1069
direc       817
indir      1074
used      47872
last      64998
free      15901
```

4.3.3 Type-177 Error. The error indicator value of 177, 176, 175, etc., indicate that an inode is referenced as an entry by more directories than its actual link count indicates. This problem, similar to the type above, is serious, and should be corrected immediately. An example of this error type is shown below:

```
# check /dev/rp32
/dev/rp32:
   612   177
spcl         76
files       288
large       101
direc        18
indir       101
used       3236
last       4397
free       1098
```

The correction procedure requires an adjustment in the inode's link-count field. The exact amount is the difference between the constant, 200_8 and the inode's error-indicator. In this particular case, the difference between the two values is 1 (200_8 - 177_8). The value 200_8 represents a file having no abnormal characteristics. The error-indicator 177 represents the present status of the file. The most logical action is to *increase* the link-count field by a value equal to the difference between these two numbers. The procedure for altering the link-count field is as follows:

```
# fsdb /dev/rp32
612i
i#:  612  md: ad----rwxr-xr-x  ln: 1  uid: 0  gid: 0  s0:  0  s1:  32
a0:  891  a1:  0  a2:  0  a3:  0  a4:  0  a5:  0  a6:  0  a7:  0
at: Wed Mar 16 13:36:42 1977  mt: Sat Feb 19 17:39:55 1977
ln= 2
q
# check /dev/rp32
/dev/rp32:
spcl        76
files      288
large      101
direc       18
indir      101
used      3236
last      4397
free      1098
```

The above procedure first seeks to the starting address of the control block for i-number 612, then sets the link-count field to 2 (1 more than it was) and finally re-checks the file system to insure the procedure worked correctly.

4.3.4 Type-201 Error. The last type of error diagnostic consists of the error indicator 201. This type of error implies that a directory entry is "missing"; it is *not* serious and does *not* require immediate attention. However, the data blocks associated with the inode will remain unavailable until the problem is solved. The "missing" directory entry can be either an *ordinary file* or a *directory*. Therefore, the first step involves identifying the i-number as either a directory or an ordinary file. This identification process is accomplished by examining the mode field in the inode structure for the particular i-number. If the inode is identified as a directory, its repair procedure is deferred until the end of this section. Otherwise, the inode is an ordinary file and its data blocks can be salvaged by copying the inode's i-number in an empty entry in some directory. The directory which is involved in this process is called "201dir"; it resides as a top-level directory under the root directory of each file system on all of the PWB/UNIX Systems. An example of this error type is shown below:

```
# check /dev/rp0
/dev/rp0:
   427  201
   431  201
spcl        174
files       264
large        92
direc        29
indir        92
used       2903
last       6681
free       3650
```

The repair process for this error type is different from all the previous procedures because a single-user environment is not required. The repair procedure is as follows:

1. The initial step is to identify the i-number of the directory "201dir":

```
# ls -id /201dir
92 /201dir
# chdir /201dir
```

2. We are now able to locate ourselves at that position in the *ilist* of the root file system:

```
# fsdb /dev/rp0
92i
#i: 92 md: ad----rwxr-xr-x ln: 2 uid: 0 gid: 0 s0: 0 s1: 112

a0: 2480 a1: 0 a2: 0 a3: 0 a4: 0 a5: 0 a6: 0 a7: 0
at: Wed Mar 30 11:57:12 1977  mt: Mon Feb 28 07:40:09 1977
```

3. The next step is to position ourselves in the file system at address addr[0] in the 201dir's inode or control block structure. The value of addr[0] is indicated by the "•" in the above inode structure.

```
a0b.p4d
d0:  92  .
d1:   1  ..
d2:   0  446
d3:   0  x1
```

The "a0b" is a mnemonic notation for addr[0]. The "p4d" prints 4 entries in the "201dir" directory. The decimal number 92 is the i-number of the "201dir" directory. The "." represents the current directory (the result of step 1). The ".." represents the parent directory of "201dir" its i-number is 1, which represents the "root" directory or "/". The remaining two entries (d2 and d3) are currently null. Their names are *446* and *x1*, respectively. Notice their i-numbers are both zero and currently the files themselves are non-existent. These two empty entries, 446 and x1, will soon contain the i-numbers of the two 201 errors.

```
a0b.d2= 427
d2:   427
```

4. We assign the i-number 427 to the null entry 446, which now becomes the name of that *orphan*. Its full path name is /201dir/446. We repeat the procedure for the *orphan* i-number 431:

```
a0b.d3= 431
d3:   431
```

5. We now terminate the command invoked in step 2 and identify the respective owners and sizes of the two files /201dir/446 and /201dir/x1, and restore them to their owners.

```
# ls -l 446 x1
-rw-------1 deem    3072 Feb 9 08:38 446
-rw-------1 atrs    3072 Feb 9 08:37  x1
```

The name field of a directory entry, other than "." and "..", can be changed to represent any character string. The string is limited to a maximum of 14 characters. For instance, the present name of directory entry d2 in /201dir is 446; the following step will change the name from 446 to orphan446:

```
a0b.d2.nm = orphan446
```

The procedure in steps 1 through 5 above assumes the presence of empty (null) entries in the "201dir" directory. If these empty entires are not present, the user must then make them available before steps 4 and 5 can be executed. Step 3 notifies the user of the existence of such null entries by printing their names. If they do not appear, the following steps will create these missing null entries:

```
q
# cp /dev/null x
# cp x x1
# cp x1 x2
# rm x x1 x2
```

The above terminates the *fsdb* command of step 2 and creates three empty files in 201dir, (x, x1, x2), which are then immediately removed consequently producing the null entries in 201dir. The repair procedure can now be restarted at step 2. This procedure is not complex but is very susceptible to human mistakes; therefore, it is necessary that each step be throughly understood.

The affected i-number of a "type-201" error, as well as the previous error types (177, 100, 377, etc), can apply to a *directory*, a *special-file*, or *no file at all*, in addition to an *ordinary file*. The various types of

PWB/UNIX files and their description are given in [1]. The first word in the i-number's inode structure identifies the file type and indicates inode allocation and file permissions:

file type:	
directory	d
character device	c
block device	b
large file	l
ordinary file	-
inode allocation	a
file permissions:	
owner	rwx
group	rwx
others	rwx

e.g.,

```
fsdb /dev/rp0
15i
i#: 15 md: ad----rwxr-xr-x ln: 2 uid: 11 gid: 1 s0: 0 s1: 3050
a0: 660 a1: 671 a2: 724 a3: 824 a4: 34 a5: 1056 a6: 0 a7:  0
at: Thu May 12 16:05:10 1977 mt: Wed May 25 10:53:11 1977
q
```

When a "type-201" error is identified as a directory, the repair process is different than for an ordinary file. The procedure is similar to that for resolving a "177" error diagnostic. The directory's link count must be *decreased* by a value equal to the difference between the error-indicator value (201_8) and the constant 200_8, namely by 1.

5. ACKNOWLEDGEMENTS

I wish to thank Rich Graveman and Larry Wehr for suggestions on clarifying and simplifying some of the more complex procedures. I would also like to express my gratitude to T. A. Dolotta for reviewing several drafts of this document and to Margarita Rivera for her help in its preparation.

6. REFERENCES

[1] D. M. Ritchie and K. Thompson, The UNIX Time-Sharing System, *Comm. ACM* 17(7):365-75 (July 1974).

[2] T. A. Dolotta, R. C. Haight, and E. M. Piskorik, eds., *PWB/UNIX User's Manual—Edition 1.0*, Bell Laboratories (May 1977).