1302
UNDS

Bell Laboratories

subject: Exercises in Repairing PWB/UNIX File Systems
File: 39382-900

date: October 19, 1978

from: P. D. Wandzilak
PY 9442
1A-114 x7344

MEMORANDUM FOR FILE

1. INTRODUCTION

This document describes a tool that generates a series of self-instructional exercises based on the concepts presented in reference [3]. The tool is an interactive PWB/UNIX* program that simulates the various degrees of file structure damage that are presently reported by the check(VIII) command. The workshop exercises are intended to provide users with the necessary framework that will enable them to become more proficient in repairing a damaged PWB/UNIX file system.

I assume that the reader has an understanding of the descriptions in references [1], [2], and [3]. I also recommend that copies of this document and reference [3] be periodically consulted for detailed explanations and procedures on resolving the more complicated exercises that might be generated in using this tool.

2. BASIC CAPABILITIES

Fsrepair is an interactive PWB/UNIX program that selectively alters the structure of a consistent file system to contain one or more of the various degrees of file structure damage that are currently reported by check. The tool allows the individual to concentrate on the problem area(s) to which he or she needs more exposure. Figure 1 contains a complete list of all the file system damage that is generated by this program.

A. Block Diagnostics

Mnemonic Identifier	Description
bad	The block number contains a value outside the allowable space on the file system.

* UNIX is a Trademark of Bell Laboratories.

dup	The block number is assigned to a previously encountered file.
din	The block number references a block of directory entries that contains an i-number that resides outside the range of the ilist size of the file system. All inodes must reside in the portion of the file system identified as the ilist.

B. Inode Diagnostics

Mnemonic Identifier	Description
100	Usually references an allocated inode which contains a zero value for its link count. An inode is a 32-byte structure that identifies various attributes of each file or directory (See File System(V)).
177	The inode is an entry in one or more directories than indicated by its actual link count references.
201	The inode is missing a directory entry.
377	An unallocated inode is claimed as an entry in a directory.

C. Free-list Diagnostics

Mnemonic Identifier	Description
dups in free	Identifies any duplication of disk blocks.
Bad Free Block	A block number in the free list lies outside of the range of available space.
missing block(s)	Identifies those block(s) which are neither allocated nor immediately available for allocation.

Figure 1.

All of the above error diagnostics are generated by the program fsrepair. However, only those error diagnostics identified as

block diagnostics (i.e., list A) or inode diagnostics (i.e., list B) may be selected as exercises. The free-list diagnostics are only generated as the result of constructing an exercise selected from either the block diagnostics or inode diagnostics. The free-list diagnostics can not be selected as individual exercises.

3. GETTING STARTED - DISK PACK INITIALIZATION

This section must be throughly read and understood before invoking fsrepair. The implementation of this program requires a spare disk drive and a minimum of two file systems of moderate sizes (e.g., copies of the /usr file system). Only one of the two file systems will be acted upon by this program. The remaining file system provides a backup, which can be used to either reinstate the active file system to a consistent state (beginning stage), or it can provide an effective means for resolving the more complicated disk damage on the active file system (e.g., simulating a spare disk drive).

A directory must be provided, containing the following executable commands:

a. mountme

b. umountme

c. removeme

d. fsrepair

The executable commands in a., b., and c. above are intended to be used in place of the mount(VIII), umount(VIII) and rm(I) commands, respectively. These commands are owned and executed by the root login. The user is advised to be aware of these conditions and to use caution when using one of these commands. The command referenced in d. above is the name of this program which the user will invoke once these initialization procedures have been performed. This command, unlike the others, is not owned or executed as root. The following steps summarize the various functions, and the order in which they should be performed, in order to establish a working environment that is a basis for the operation of this program. The procedural steps are as follows:

1. Obtain a scratch disk pack that has been recently formatted.

2. Physically mount this disk pack on a spare disk drive. Next copy (see vc(VIII)) a non-root file system (e.g., /usr) onto this disk pack in two different locations.

3. Re-label (see labelit(VIII)) the two file system copies with their appropriate volume serial numbers and file system identification names as follows:

```
labelit /dev/rpXX  [fsname] [vol-ser. #]     first copy
labelit /dev/rpYY  [fsname] [vol-ser. #].    second copy
```

4. Change the owner of both special files, character and block corresponding to each of these two file systems (e.g., /dev/rpXX and /dev/rrpXX). The new owner must correspond with the owner of the fsrepair program. Also, the modes (permissions of the file) for both special files should be read and write by this owner (see chmod(I)).

5. Check the two file systems on the scratch disk pack to insure that each file system is consistent. This is a crucial step because if file damage is already present, invoking the fsrepair program on such a file system can cause unexpected results.

6. Change the ".path" file in the user's login directory to allow for an alternate directory search, which includes the directory, /etc. This provides a more convenient method for allowing the user to access and execute commands that would normally require a fully qualified path name (e.g., check, clrm, icheck, etc.). Figure 6 represents an example which illustrates the format for this new directory search.

```
:/etc:/bin:/usr/bin
```

Figure 6.

However, when the ".path" file is damaged, the user is required to first terminate the process for his/her current login shell (i.e., log off the system) and then invoke a new login shell process (i.e., re-log onto the system) before invoking this program. This forces the user's .path file to be re-read and allows the changes that were made to this file to become effective.

7. Invoke the fsrepair program.

4. PROGRAM USAGE

The program is designed to be invoked on a moderate-sized PWB/UNIX file system which has an adequate supply of files (large and small), directories, and non-allocated inodes (for example, the contents of the /usr file system). The program operates on the basis of the availablity of these resources. Once a resource or any combination of these resources become unavailable, a notification in the form of an ERROR diagnostic is issued, and

the program terminates. Section 5 describes the various conditions which generate ERROR diagnostics.

4.1 Initializing an Error Condition

Invoking the program with a list of arguments enables the users to choose only those error diagnostics from the lists in either A or B in Figure 1, which are of interest to them. Each argument is named by their respective mnemonic identifiers (e.g., dup, din, 100, ..., etc.) and successive arguments are separated by one or more blank spaces. Figure 2 represents an example which illustrates this format:

```
% fsrepair bad 377 ... dup ... etc. (return key)
```

Figure 2.

The program then processes each argument in the list sequentially. Once the last argument in the list has been processed, the user can request additional exercises by choosing a particular error diagnostic and entering its corresponding mnemonic identifier through the terminal. Refer to Section 4.3.

In addition to Figure 2, the program can also be invoked without any arguments. In this mode of operation, the user is then temporary restricted from selecting his or her own exercises. This restriction is enforced until an exercise for each error diagnostic from both lists A and B in Figure 1 have been processed, after which the user can request additional exercises of his or her own choice in the same manner that was previously discussed. Figure 3, represents an example which illustrates this format.

```
% fsrepair (return key)
```

Figure 3.

4.2 Interactive Use

Once the program is invoked, the remainder of the session is then conducted interactively. The program solicits a series of prompt messages on the user's terminal. These prompt messages are organized into two categories. The first category of prompts require the user to reply with an appropriate response. For example, the program will prompt the user's terminal with the following message:

```
ENTER DEVICE NAME (e.g., 24, 14, 44):
```

This message requires the user to identify the device name for the file system where these exercises will be constructed by entering only the minor device number associated with the special

file, /dev/rp??. The minor device number specifies both the physical drive unit and a designated logical portion of the device (e.g., 24 - The physical drive unit is represented by the first digit, 2, and the succeeding digit, 4, represents the designated logical portion of the device). All messages in this category inform the user on the kind of response that is required.

The remaining prompt category provides the user with a direct connection to the PWB/UNIX shell, via the -c flag. The user is notified of this mode of processing by the special prompt character "->" appearing on his or her terminal. The remainder of the line after the prompt character is then sent to the UNIX shell (sh(I)) to be interpreted as a legal command. The commands "chdir" and "cd", which are invoked directly by the shell, will not change the user's current directory, ".", when invoked under these conditions. The user must supply the fully qualified path name when accessing commands that require a directory search sequence which is different than either the default value, :/bin:/usr/bin, or any alternate search sequences that are specified in the file named ".path" in the user's login directory.

The program remains in this mode of processing until either a "break" of "del" character is typed by the user. The program preempts all processing in this mode, and prints a message on the user's terminal requesting the user to direct the program on what to do next. Figure 4 illustrates this sequence of events.

```
(a)     -> /etc/check /dev/rrp22    (return key)
        /dev/rrp22:
        spcl       84
        files     340
        large     102
        direc      32
        indir     102
        used     3237
        last     4060
        free     1068

(b)     -> ("del" or "break" key)

TYPE S TO (STOP), N FOR (NEW EXERCISE) OR "RETURN KEY" TO CONTINUE:
```

where:

S, or s	Terminates the program and control returns at the shell level (i.e., %).
N, or n	Produces a new exercise after prompting the user to select the desired exercise (e.g., 100, 201, bad, ...).
Return Key	Control returns to the prompt mode before the interrupt (del or break key) occurred (i.e.,

->; connection to the PWB/UNIX shell).

Figure 4.

4.3 Selecting an Error Condition

Step (a), in Figure 4 illustrates this facility of directly accessing UNIX commands through a direct connection to the PWB/UNIX shell. This convention remains intact until the user initiates an interrupt as shown in step (b) above. The user is then free to choose other exercises. Figure 5 represents an example which illustrates this process of selecting another exercise.

(a) -> ("del" or "break" key)

(b) TYPE S TO (STOP), N FOR (NEW EXERCISE) OR "RETURN KEY" TO CONTINUE: N

(c) ENTER ONE EXERCISE ONLY (e.g., 100, 201, bad, ..., etc.): 377

EXERCISE SELECTED ** 377 **

(d) ->

where:

(a) The "del" or "break" character disconnects the connection and allows the user to alter and control the flow of processing.

(b) Request for a new exercise. Both upper or lower case ASCII characters are accepted (e.g., N or n).

(c) The user selects the desired exercise by entering the corresponding mnemonic identifier for the exercise.

(d) The desired exercise is now ready and the connection to the shell is again reinstated.

Figure 5.

4.4 Repairing an Error Condition

Once an exercise has been selected and the connection to the PWB/UNIX shell is established (i.e., ->), the user can then proceed with the necessary repair steps to resolve the particular error condition. The appropriate section of reference [3] describes specific procedures designed to resolve the particular error condition. An example of a procedure for repairing an error condition is as follows:

```
(a)     -> /etc/check /dev/rrp14
        /dev/rrp14:
           17    100
          519    100
        spcl         0
        files    11364
        large     1069
        direc      817
        indir     1074
        used     47872
        last     64998
        free     15901

(b)     -> /etc/clrm /dev/rrp14 17 519

(c)     -> /etc/icheck -s4 /dev/rrp14
        /dev/rrp14:

(d)     -> /etc/check /dev/rrp14
        /dev/rrp14:
        spcl         0
        files    11362
        large     1069
        direc      817
        indir     1074
        used     47872
        last     64998
        free     15915
(e)     ->
```

Only after an exercise is completed (i.e., the error condition is repaired) should the user select another exercise as described in section 4.3.

5. PROGRAM DIAGNOSTICS

This section describes the various conditions that cause an internal error message to be generated and the solutions for resolving each of these conditions. The error messages currently exist in two categories, fatal errors (ERROR:#:message body), which are followed by program termination, and non-fatal errors (WARNING:#:message body), which do not terminate the program. All messages in both categories conform to the following format specification:

Category:Number:Message Body

where:

Category	identifies the message category as either fatal or non-fatal.
Number	identifies the error condition.
Message Body	explains briefly the error condition.

5.1 Fatal Messages

A fatal condition refers to an occurrence of a significant error within the program that cannot be self-satisfied, but rather requires external methods to resolve the pending condition. A message printed as a result of these conditions are called ERROR messages and are immediately followed by the termination of the program. The user or a PWB/UNIX Administrator is then expected to remedy the problem before fsrepair can be re-invoked. Once the error condition has been corrected, and the active file system is restored to a consistent state, fsrepair can be re-invoked.

The individual messages included in this category as well as convenient methods for resolving them are discussed below.

ERROR:1: STAT ERROR

This message is printed if either the root directory (/) or the special file (/dev/rp??) supplied by the user is not accessible. This message occurs when the file could not be located or the directory names in the file's path name were not accessible.

SOLUTION: Check if the special file exists or for an improper setting of the file's permission bits (mode, ownership). Refer to Section 3.

ERROR:2: ROOT DEVICE

This message occurs when an attempt is made to invoke this program on the root file system.

SOLUTION: Re-invoke the program using a non-root file system.

ERROR:3: FILE NOT SPECIAL

This message indicates that the selected file does exist in directory /dev. However, the selected file is not a special file (character or block device).

SOLUTION: Re-invoke the program with a proper block device name (e.g., /dev/rp44).

ERROR:4: CAN NOT READ AND WRITE THE BLOCK DEVICE

The special file, /dev/rp??, can not be read or written by the owner or has an owner that is different than the current owner of this program.

SOLUTION: Reset the permission bits in the mode field so the owner can read and write the file. Also, change the owner of the file to have the same owner as this program.

ERROR:5: CAN NOT READ THE SUPER-BLOCK

This message occurs when the read(II) syscall issued on block one of the active file system terminates prematurely. This message can be caused by the device of the file system being off-line or by an imperfection on the recording surfaces of the disk pack.

SOLUTION: If the device of the file system is off line, turn the power on and then re-invoke the program. Otherwise, select a second disk pack and perform the steps in Section 3 before re-invoking this program.

ERROR:6: FILE SYSTEM SIZE CHECK

The super-block of the file system contains erroneous information for either the isize or fsize parameter specifications. The variable, isize, the first word entry in the super-block of the file system, contains the block size of the ilist region. The variable, fsize, the second word entry in the super-block, contains the block size of the entire file system. This message can be attributed to the file system not having a valid PWB/UNIX file system structure. This should not happen, unless the procedure

steps in Section 3 were not performed correctly, or an error occurred while these steps were being performed.

SOLUTION: Recopy the super-block (block one) using the command, bcopy(VIII) or re-execute the procedure steps in Section 3.

ERROR:7: CAN NOT ALLOCATE ENOUGH CORE

This message indicates that the syscall alloc(III) could not succeed in granting the amount of additional core memory that was requested. Presently, this syscall is called only once and requests for the following additional core memory:

(isize * 16 + 1).

The additional core memory is used by the program for determining which inodes will not be used in implementing the current exercise. The error occurred because the combined text and data space exceeded the maximum boundary limitation for the user's current virtual address space.

SOLUTION: Re-compile the program with separate I/D space. This separates the program's text (instructions) from the data space and allows for greater expansion in both of these regions. Refer to cc(I) in reference [2] for complete instructions on the C compiler.

ERROR:8: NO REMAINING FILES

This message indicates that the current exercise failed, because there are no available file(s) remaining in the file system in which to construct the exercise.

SOLUTION: The remedy is to re-invoke the program, but only if the file system is consistent.

ERROR:9: NO REMAINING DIRECTORIES

This message indicates that the current exercise failed, because there are no available directories remaining in the file system in which to construct the exercise.

SOLUTION: Same as for ERROR # 8.

ERROR:10: NO REMAINING FREE INODES

This message occurs when an attempt to allocate a free inode is made and fails. The failure resulted because there are no free inodes remaining in the file system.

SOLUTION: The remedy is to copy the backup file system (See Section 3) onto the file system and then re-invoke the program.

5.2 Non-Fatal Messages

A non-fatal condition refers to those errors that do not warrant program termination. A message printed as a result of these conditions are called "WARNING" messages. In general, messages in this category pertain to invalid prompt responses by the user and to an occasional error in reading or writing a disk block.

WARNING:1: INVALID ARGUMENT - SKIPPING TO NEXT ARGUMENT

The message indicates that the user selected an exercise which is not supported by this program. Both lists A and B in Figure 1, include all the exercises which are currently supported by this program.

SOLUTION: Choose an exercise from either list A or B in Figure 1.

WARNING:2: CAN NOT READ A DISK BLOCK

The message indicates that the read(II) syscall failed for the current operation on device (/dev/rp??) at location (block #). The exercise is then terminated.

SOLUTION: Re-select the exercise that provoked the error. If the error condition continues to occur, terminate the program and consult with the PWB/UNIX System Administrator.

WARNING:3: CAN NOT WRITE A DISK BLOCK

The write(II) syscall failed for the current operation on device (/dev/rp??) at location (block #). The exercise is then terminated.

SOLUTION: Refer to the description given for WARNING # 2.

WARNING:4: FILE SYSTEM IS MOUNTED

Fsrepair checks for this condition when it is initially invoked. This message reminds the user that the file system where these exercises will be conducted is already a mounted file system and that the file system must be unmounted before it can be remounted.

SOLUTION: No action is required.

WARNING:5: INVALID INSTRUCTION - TRY AGAIN

The message indicates that an invalid prompt response was received. All prompt messages in this category inform the user which prompt responses are acceptable. A response that is different than what is expected is then disregarded and the prompt is re-issued.

SOLUTION: Read the prompt message carefully and then decide on the proper response.

WARNING:6: NO REMAINING BLOCKS ON FREE LIST

The message occurred when the in-core free list has been

exhausted. The exercise, not the program, is then terminated at the point where the error condition occurred.

SOLUTION: Reconstruct the free list by invoking the command icheck(VIII). Reselect the exercise that failed as a result of this error condition.

6. AN ILLUSTRATIVE EXAMPLE

Figure 7 represents an example which contains the initial invocation of the program, selection of exercises, generation of various program errors, and the termination of the program.

A. % fsrepair 100

B. INFORMATION (y/n)? n

C. ENTER DEVICE NAME (e.g., 24, 14, 44): 22

*** PROGRAM INITIATED ***

DEVICE: /dev/rp22

FILE NAME: class
VOLUME NAME: adm

D. -> /etc/check /dev/rrp22
/dev/rrp22:
17 100
519 100
spcl 84
files 339
large 102
direc 32
indir 103
used 3488
last 4060
free 817

E. -> ("del" or "break" key)

F. TYPE S TO (STOP), N FOR (NEW EXERCISE) OR "RETURN KEY" TO CONTINUE: n

G. ENTER ONE EXERCISE ONLY (e.g., 100, bad, etc.): 177

EXERCISE SELECTED ** 177 **

H. -> /etc/check /dev/rrp22
/dev/rrp22:

```
   17   100
   95   177
   96   177
  519   100
spcl      84
files    339
large    102
direc     32
indir    103
used    3488
last    4060
free     817
```

I. -> ("del" or "break" key)

J. TYPE S TO (STOP), N FOR (NEW EXERCISE) OR "RETURN KEY" TO CONTINUE: 20

WARNING: 5: INVALID INSTRUCTION - TRY AGAIN

K. TYPE S TO (STOP), N FOR (NEW EXERCISE) OR "RETURN KEY" TO CONTINUE: n

L. ENTER ONE EXERCISE ONLY (e.g., 100, 201, bad, ..., etc.): 202

WARNING: 1: ILLEGAL ARGUMENT - SKIPPING TO NEXT ARGUMENT

M. ENTER ONE EXERCISE ONLY (e.g., 100, 201, bad, ..., etc.): din

EXERCISE SELECTED ** din **

N.
```
-> /etc/check /dev/rp22
/dev/rrp22:
1324 din: inode = 98 (data(small))
1324 din: inode = 98 (data(small))
1324 din: inode = 98 (data(small))
1324 din: inode = 98 (data(small))
1 dups in free
5 missing
   17   100
   95   177
   96   177
   98   201
  101   201
  519   100
spcl      84
files    339
large    102
direc     32
indir    101
used    3488
last    4060
free     817
```

O. -> ("del" or "break" key)

```
P.    TYPE S TO (STOP), N FOR (NEW EXERCISE) OR "RETURN KEY" TO CONTINUE:  S

                *** PROGRAM TERMINATED NORMALLY ***
```

Figure 7.

where:

A The program is invoked with one argument (100).

B Names of documents that are associated with this tool.

C Select the file system where the session will be conducted.

D Facility to access and execute commands via the PWB/UNIX shell.

E An interrupt ("del" or "break") signal was detected and step D is then terminated.

F Select the next action for the program to do next. This message occurred because of the action taken in step E.

G This message occurs when the user chooses to select a new exercise, as in step F.

H Facility for accessing and executing UNIX commands is reinstated.

I Same as in step E above.

J An invalid instruction (202) was given and results in an error condition (see Section 5).

K The prompt message will continue to occur until a correct response is supplied.

L An improper exercise was selected. This results in an error condition. Refer to lists A or B in Figure 1 for the names of all exercises that are supported by this program.

M This prompt message is re-issued due to the error condition in step L. A proper exercise was then selected.

N The facility for accessing and executing UNIX commands is once again reinstated.

O An interrupt signal was detected and step N is immediately terminated.

P Due to the interrupt, this prompt message was produced. The program is terminated.

7. CONCLUSION

File system integrity is an integral part of installing, operating, and administering a PWB/UNIX system. Therefore, every effort must be made to insure that those persons responsible for maintaining a PWB/UNIX system have adequate knowledge and training in this area. It was with this intention in mind that this tool was developed. This tool is presently being used in the course entitled, "Operating, Installing and Administering a PWB/UNIX System", as a workshop exercise for training Bell System Customers, and to train PWB/UNIX operations personnel.

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, PWB/UNIX User's Manual - Edition 1.0, Bell Laboratories (May 1977).

[3] P. D. Wandzilak, Repairing Damaged PWB/UNIX File Systems, Bell Laboratories (March 1978).

P. D. Wandzilak

P. D. Wandzilak

PY-9442-PDW-pdw

Copy to
A. P. Boysen, Jr.
M. P. Fabisch
9442 PWB Development Group
9442 PWB System Engineering Group
9442 Supervision