UNOS 1555

Bell Laboratories Cover Sheet for Technical Memorandum

The information contained herein is for the use of employees of Bell Laboratories and is not for publication (see GEI 13.9-3)

Title: **On Making the UNIX™ File System Crash Resistant** Date: **May 27, 1980**

Other Keywords: **recovery**
unattended operation TM: **80-3168-8**

Author(s)	Location	Extension	Charging Case: 49408-120
Alan L. Glasser	**HO 1E-335**	**6569**	Filing Case: **40324-2**

ABSTRACT

Changes have been made to the UNIX kernel that make its file system crash resistant. A system running this modified kernel has a higher degree of availability than a standard UNIX system. Only a very few routines in the kernel required modification; only 30 lines of code were changed. In addition to the kernel changes, a *salvage* program was written to recover disk space after a crash. These changes do not include any optimizations; some possible optimizations are presented. With worst case test programs (i.e., programs that are *write* intensive), this implementation is four times slower than the standard system. The modifications described here do *not* affect read performance. For a more realistic read/write application, the cost of using this implementation may well be less than a factor of two.

Pages Text: 4	Other: 0	Total: 4
No. Figures: 0	No. Tables: 0	No. Refs.: 6

TM-80-3168-8

SCHEIDERMAN,CAROLE L CM
MH2F128 06/04/80
SUBJECT MATCH UNCS

E-1932-U (3-76) SEE REVERSE SIDE FOR DISTRIBUTION LIST

BELL TELEPHONE LABORATORIES, INC.

TM-80-3160-8

DISTRIBUTION
(REFER GEI 13.9-3)

COMPLETE MEMORANDUM TO

CORRESPONDENCE FILES

OFFICIAL FILE COPY
PLUS ONE COPY FOR
EACH ADDITIONAL FILING
CASE REFERENCED

DATE FILE COPY
(FORM E-1328)

10 REFERENCE COPIES

AAGESEN,JOHN
ABATEMARCO,TERESA N
ABATE,JOSEPH
AITCHESON,E J
ALBAGLI,V E
ALBERTS,BARBARA A
ALKONS,FREDERICK
AMABILE,GEORGE E
ANDERSON,C E
ANDREWS,E J
ANDREWS,W J
ARCHER,RUSSELL E,JR
ARMAN,THOMAS D
ARMSTRONG,D E
ARNOLD,THOMAS P
ASPIN,LEE J
ATTIS,H P
ARVIDSON,W P
BACCASH,JEANNE M
BACH,MAURICE J
BAKER,DONN
BALENSON,CHRISTINE M
BARNHARDT,KARL E
BABOFSKY,ALLEN
BARON,ROBERT V
BARR,DAVID L
BARR,W J
BARTON,M E
BAUER,BARBARA T
BAUER,H C
BAUER,WOLFGANG P
BAYZER,RICHARD J
BAYER,D L
BEAUMONT,LELAND E
BECKETT,J T
BENISCH,JEAN
BERGESON,E F,JR
BERKOWITZ,PAUL E
BERNHARDT,RICHARD C
BERNOSKE,BEVERLY G
BERRYMAN,E D
BIREN,EMMA B
BISCHOFF,B BARRY
BITTRICH,MARY E
BLAHUT,D E
BLAZIER,S D
BLEIER,JOSEF
BLINN,J C
BLOSSER,PATRICK A

COMPLETE MEMORANDUM TO

BLUM,MARION
BOCKUS,ROBERT J
BOEHM,EARL W
BOGART,THOMAS G
BOHNING,J E
BOIVIE,RICHARD H
BOLINSKI,NANCY V DEVLIN
BONACHEA,E N
BOSTON,RONALD E
BOURNE,STEPHEN R
BOWYER,L RAY
BOYCE,K J
BOYCE,W M
BRADFORD,EDWARD G
BRADLEY,M HELEN
BRANDT,RICHARD B
BRAUN,L E
BRAUN,DAVID A
BRAUN,E J
BREITHAUPT,A A
BRENSKI,EDWIN P
BRITT,WARREN D
BROSS,JEFFREY D
BROWN,ELLINGTON L
BROWN,JAMES M
BROWN,W E
BURNET,ROSE M
BURROWS,THOMAS A
BUTLETT,DARRELL L
BUZZARD,C ALAN
BYORICK,ROBERT S
CAMPBELL,JERRY H
CANADAY,RUDD H
CARBAN,JOHN H
CARK,DAVID C
CARTER,DONALD H
CERNAK,I A
CHAPPEE,M P
CHAI,D T
CHANG,JO-MEI
CHANG,S-J
CHAO,C
CHAPPELL,S G
CHARLES,JOYCE
CHELLIS,ALICIA L
CHEN,ROBERT
CHEN,STEPHEN
CHEN,T L
CHERRY,LORINDA L
CHIANG,T C
CHODROW,M M
CHONG,PHEE
CHOW,W F
CHO,YUN-CHUNG ERIC
CHRISTENSON,DENNIS L
CICHINSKI,STEVEN
CICON,J P
CIRILLO,C
CLIFFORD,COURTENAY S
CLINE,LAUREL M I
CLINE,TERRY W

COMPLETE MEMORANDUM TO

COHEN,AARON S
COHEN,DAVID
COLEMAN,C L
COLEMAN,ELANE
COLLICUTT,A B
COMMRADT,GERALDINE
COOK,T J
COPP,DAVID H
COREY,D A
COSTELLO,PETER E
COWELL,ARLINE D
COYNE,DENNIS C
CRACOVIA,E V
CRANE,E F
CRISTOFOR,EUGENE
CRONE,MARGARET M
CSASZAR,MARIANN
CSURI,JOHN O
DALE,O B
DALRYMPLE,FREDERICK L
DAVIDSON,CHARLES LEWIS
DAVIDSON,ROBERT P
DE GRAAF,D A
DE JAGER,D S
DE LOGISH,B G
DE NIGRIS,ERNEST G
DEAN,JEFFREY S
DENNI,MICHAEL S
DICKMAN,BERNARD N
DIMMICK,JAMES O
DIVAKARUNI,R S
DIXON,DAVID A
DOBLMAIER,A H
DOCK,G A
DOLE,E F
DOLOTTA,T A
DOMPIERRE,J A
DONNELLY,MARGARET M
DONOHOE,D C
DONOHUE,E F,3RD
DOUGHERTY,E J
DOUDEN,DOUGLAS C
DOWD,PATRICK G
DRUMMOND,R E
DUBMAN,M E
DUCHARME,ROBERT LAWRENCE
DUNLOP,ALFRED E
DVORAK,P S
DWYER,T J
EDMUNDS,T W
EITELBACH,DAVID L
ELDREDGE,GARY P
ELMENDORF,C H
ELY,T C
EPLEY,ROBERT V
ERICKSON,VERLYN E
ERRICHIELLO,PHILIP M
ERWIN,E J
EVENSON,E K
FABRICIUS,WAYNE M
FACTOR,ROBERT M

COMPLETE MEMORANDUM TO

FASCIANO,V
FAVIN,D L
FEDER,J
FEINBERG,HENRY E
FEINER,A
FENG,FRANK E
FINGER,ERIC J
FINK,BERNICE A
FISCHER,HERBERT E
FISHER,W A
FISHMAN,DANIEL H
FITTON,MICHAEL J
FLANDRENA,E
FLEMING,JAMES E
FOLEY,G
FONG,K T
FORMICA,STEPHEN J
FORTNEY,F J
FOUGHT,E T
FOUNTOUKIDIS,A
FOY,J C
FRANKLIN,C M
FRASER,A G
FREDERICKS,A A
FREEMAN,K G
FREEMAN,MARTIN
FREEMAN,R DON
FRITZ,THOMAS E
FROST,H BONNELL
FU,C
GALLANT,E J
GANGAWARE,BERNICE C
GARST,BLAINE,JR
GATES,G W
GEORGEN,MICHAEL E
GEPNER,JAMES E
GERARD,A
GERRISH,A M
GEYLING,F T
GEYNET,ROSEMARY
GLAB,KENNETH E
GLSON,E T,JR
GILKEY,THOMAS J
GILLON,ALEX C
GIMMELLI,RALPH T
GINGRICH,PATRICIA S
GLASSER,ALAN L
GLAZER,STANLEY
GOFF,CAROLYN E
GOGUEN,N E
GOLD,RHONDA L
GOODMAN,JESSE D
GORDON,MOSHE E
GRAHAM,E L
GRAVEMAN,E F
GREENBAUM,HOWARD J
GRESHAM,J P
GRIFFITH,KIMBERLEY ANN
GRIFFITH,E J
GROSS,ARTHUR G
GUIDI,PIER V

COMPLETE MEMORANDUM TO

HAASE,JOHN L,3RD
HACHENBURG,V
HAHN,JAMES E,JR
HAIGHT,E C
HAISCH,H F,JR
HALLER,N M
HALLIN,THOMAS G
HALL,ANDREW D,JR
HALL,WILLIAM G
HALPIN,T
HAMES,ROSALYN
HAMILTON,LINDA L
HAMM,DEBORAH ANN
HANSEN,GWEN J
HANSEN,E C
HANSON,BRUCE L
HARKNESS,CAROL J
HARRISON,JAMES FRANCIS
HARRIS,BRENDA L
HARRIS,WARD A
HARTMANN,E H
HARTOIN,ROBERT E
HARVEY,D E
>HAYDEN,DONALD F,JR
HAYES,MARILYN E
HAYWARD,HENRIETTA M
HEATER,ROBERT J
HEDRICK,ELLEN L
HEFFRON,W GORDON,JR
HEIDER,BRUCE E
HELD,E W
HELLER,J ALLEN
HELLER,K A
HENIG,FRANCES E
HERBST,ROBERT T
>HERGENHAN,C E
HERMAN,KENNETH M
HERNDON,JOHN A
HESSELGRAVE,MARY E
HESS,E E
HILLYARD,E E
HOFMANN,A M
HOLTMAN,JAMES P
HOLTZMAN,JACK M
HOOKER,J V
HOOVER,ERNA E
HO,DANIEL K Y
HO,DON T
HO,TIEN-LIN
HSU,TAU
HUBER,DEBORAH J
HWANG,HENRY
IMAGNA,C P
IPPOLITI,O D
IRVINE,M M
ISENMAN,M E
ISMAN,MARSHALL A
IULIANO,VINCENT P
JAASMA,E G
JABLONSKI,GRAZINA C
JACKOWSKI,D J

...
681 TOTAL

♦ NAMED BY AUTHOR > CITED AS REFERENCE < REQUESTED BY READER (NAMES WITHOUT PREFIX WERE SELECTED USING THE AUTHOR'S SUBJECT OR ORGANIZATIONAL SPECIFICATION AS GIVEN BELOW)

MERCURY SPECIFICATION.....

COMPLETE MEMO TO:
3121-SUP 3122-SUP 316-SUP 3160-TA 323-SUP 324-SUP 3392-SUP 3491-SUP 363-SUP 364-SUP
9224-SUP 932-SUP 933-SUP 9362-SUP

COCSPS = COMPUTER FILE SYSTEMS
UNOS = UNIX OPERATING SYSTEM: GENERAL OR SURVEY DOCUMENTS

COVER SHEET TO:

NO CORRESPONDENCE FILES
NO 1A127

TM-80-3160-8
TOTAL PAGES 6

TO GET A COMPLETE COPY:

1. BE SURE YOUR CORRECT ADDRESS IS GIVEN ON THE OTHER SIDE.
2. FOLD THIS SHEET IN HALF WITH THIS SIDE OUT AND STAPLE.
3. CIRCLE THE ADDRESS AT RIGHT. USE NO ENVELOPE.
4. INDICATE WHETHER MICROFICHE OR PAPER IS DESIRED.

PLEASE SEND A COMPLETE

( ) MICROFICHE COPY ( ) PAPER COPY

TO THE ADDRESS SHOWN ON THE OTHER SIDE.

Bell Laboratories

subject: **On Making the UNIX™ File System Crash Resistant**
Charge Case 49408-120
File Case 40324-2

date: **May 27, 1980**

from: **Alan L. Glasser**
HO 3168
1E-335 x6569

TM 80-3168-8

MEMORANDUM FOR FILE

1. INTRODUCTION

The UNIX time sharing system was developed to provide computer scientists with a comfortable and effective computing facility. It has been used in many other arenas, notably for operations support systems and as a programmer's workbench. Unfortunately, UNIX was not designed for the extremely high availability or unattended operation requirements of many Bell System applications. In particular, the UNIX file system implementation is optimized for performance, rather than crash resistance. This memorandum describes a small set of changes to the UNIX kernel that yield a UNIX system that can be halted at any time and its file systems subsequently mounted read-only without any intervening processing; also, full read-write capabilities are possible once a (non-interactive, non-heuristic) *salvage* program is run. We call such a file system *crash resistant.*

2. NORMAL UNIX FILE SYSTEM RECOVERY

The following quote, from [1], is a succinct description of normal UNIX file system crash recovery:

> *Repairing disks.* The first rule to keep in mind is that an addled disk should be treated gently; it shouldn't be mounted unless necessary, and if it is very valuable yet in quite bad shape, perhaps it should be dumped before trying surgery on it. This is an area where experience and informed courage count for much.

While the UNIX/TS file system check program (*fsck*) [2] is far more thorough than its predecessors, it is heuristic, and must often process highly contradictory data. One should *not*, in general, allow *fsck* to make whatever changes in the data it deems necessary. Thus, an intelligent operator is required. (*Fsck* is far superior to other file system check programs in that less skill is required to use it successfully.)

The inconsistencies in a file system after a crash result from data being written to the disk in an order different from the original logical I/O sequence [3]. There are two causes for this reordering: disk drivers usually perform head movement optimizations that change the sequence in which data is written, and the UNIX buffer cache tends to change the order in which data is presented to the disk driver.*

3. AN IMPLEMENTATION OF A CRASH RESISTANT FILE SYSTEM

3.1 Kernel Overview

Were it true that "the software is careful to perform I/O in the correct (*logical*) order" [3], all that would be necessary to make the UNIX file system crash resistant is to make all writing synchronous. Unfortunately, this is a necessary, but not sufficient, change.

The philosophy followed in making further changes was to guarantee that the disk contains consistent *inodes*, indirect blocks, and data blocks at *all* times. We allow the free list to be inconsistent, and therefore require that the free list always be reconstructed on a re-boot (see "The Salvage Program", below). Also, to allow reasonable performance, the contents of the disk are allowed to lag the true contents of the file system. The result is that the completion of a *write* system call does not guarantee that the data can be recovered if the system crashes. A *sync* system call is necessary to assure that the data is indeed recoverable. (*If* performance is not as important as timeliness, an implied *sync* can easily be added to the *write* system call.) Finally, these modifications include no optimizations. As the changes required intimate knowledge of the kernel, and as proof of crash resistance seemed difficult to obtain, simplicity was substituted for efficiency.

3.2 Kernel Details

This section describes the details of the additional changes; familiarity with the system source code (PWB 1.0) would be helpful to the reader.

Inspection of the source code revealed that during the creation of a new file (or a link to an existing file), the directory containing the file could point to an erroneous inode. This problem is avoided by adding calls to the *iupdat* routine, which writes an inode from the in-core table to the disk, in the *maknode* and *link* routines.

When creating a new file (or link), a *sync* system call is necessary to guarantee that the new file is present on the disk (this is analogous to the *write* situation described above). As file and link creation is not as frequent as writing, providing such assurance is relatively inexpensive. This guarantee is effected by adding a call to *iupdat* in the *wdir* routine.

Truncating a file, either via a *creat* or *unlink* system call, provides an opportunity for very serious file system corruption. If the system should crash during the truncation of a file, after the crash one might discover that the file had not been truncated at all, that another file claimed blocks owned by the original file (so called *dups*), and that the free list also claimed blocks owned by the original file (so called *dups in free*). This problem is avoided by rewriting the *itrunc* routine to make a copy of the inode, set the size and block pointers to zero, and write this inode back to the disk. After this, the blocks originally claimed by this file are freed via the copy of the inode.

The change to *maknode* (the added call to *iupdat*) introduced a subtle bug. The *mknod* routine (i.e., the *mknod* system call interface) calls *maknode* and subsequently modifies the inode (it sets the device number for special files). At this point the inode entry is freed. As neither the modified nor accessed bits are set (*iupdat* clears them), the inode is not re-written. The net effect is that the *mknod* system call would only make special files with device number zero. Resetting the modified bit in the inode to force a re-write is not acceptable because a crash

---

* When a process writes a block of data, the system copies that data to a system buffer, queues that buffer on the disk driver work queue, and returns control to the process. Also, when a partial block is written, the buffer is merely marked as *"write before using"*, and the physical writing deferred until the buffer is needed for other data. These techniques allow substantial overlap of CPU and disk activity, and significantly reduce the amount of physical I/O that would be otherwise required.

before the re-write leaves an erroneous inode on the disk. This bug was fixed by adding an extra argument (the device number) to *maknode* and changing all calls to *maknode* to include the additional argument.

3.3 The Salvage Program

As was stated earlier, these modifications require that, after a crash, the free list of each file system be rebuilt prior to using it. When the system runs in duplex mode** the root file system is mounted read-only. Thus the consistency of the root is guaranteed. As part of system initialization, a file system salvage process is spawned for each read/write file system that is normally mounted. The salvage program recovers all unused blocks and unreferenced inodes, adjusts link counts, builds a free list, and performs some consistency checks. The program is a heavily modified version of the old *check* program. The consistency checks are provided to help detect latent bugs in the file system code or hardware failures.

4. PERFORMANCE

These modifications yield a *write through* buffer cache, rather than the standard *write back* buffer cache, one effect of which is to increase the amount of disk traffic in the system. Also, the fact that writing is synchronous rather than asynchronous will tend to result in shorter than normal disk queues and a larger than normal amount of disk head movement. These two factors make obvious the fact that such a crash resistant file system is necessarily slower than the standard file system: performance was sacrificed for reliability.

To obtain a rough measure of the cost, a worst case test program was written that spawns a specified number of children that each create and write a file. With 10 children, each writing files of 9 blocks (a 9 block file is the smallest "large" file) repeatedly, this system is four times slower (real time) than the standard system. Also, synchronous writing makes processes wait, thereby reducing the amount of CPU/disk overlap. This results in 10% more CPU idle time that is available for other (hopefully CPU-bound) processes.

While a factor of four seems exorbitant, there are optimizations to be made. One simple, and probably significant, optimization would be to not write newly allocated (hence, empty) blocks. With proper measurement it should be possible to find other optimizations (e.g., use so-called "inverted" file systems so that pairs of file systems share a common region of the disk for their separate *ilists*—almost half the traffic of typical PWB file systems is to the *ilist*).

The test program generates atypical disk traffic as it never reads any files (hence the characterization of the program as "worst case"). The ratio of disk reads to writes for a typical PWB/UNIX system is about four. The ratio for database applications is often higher. The modifications described here do *not* affect read performance. For any given application that requires crash resistance, the cost of using this implementation may well be less than a factor of two.

5. CONCLUSIONS

Perhaps the most significant aspect of this work is that it has shown that a UNIX file system *can* be made crash resistant.

The system has been tested and the source code examined in walk-throughs. Testing was difficult (the system was deliberately stopped while various test programs were running, and

---

** The crash resistant file system changes are a small portion of the total changes made to the system. Numerous other changes were made to the system to support a dual-processor, multiple shared disk controller PDP-11/70 system designed for high availability and unattended operation [4, 5, 6].

then the file system was checked for consistency). Two avenues are available to further improve the reliability of this system: first, the software should be *soaked* for a substantial period of time, and second, an attempt should be made to prove the correctness of the modifications.

The required skill level for a craftsperson maintaining such a system is lower than that required for maintaining a standard UNIX system.

The ease of making these changes, and the minute number of changes required—less than 30 lines, out of the approximately 9000 total lines, of the kernel were changed—is another tribute to the versatility of UNIX.

REFERENCES

[1] PWB/UNIX User's Manual Release 2.0. *Crash(8)*. June 1979.

[2] T. J. Kowalski. FSCK — The UNIX/TS File System Check Program. July 1, 1979. TM 79-3624-4.

[3] K. Thompson. UNIX Implementation. *The Bell System Technical Journal* 57, 6 (July-August 1978, Part 2), pp. 1931-46.

[4] D. F. Hayden and D. M. Ungar. File Machine Recovery Plan - Revision 0. March 1, 1978. MF 78-3124-7.

[5] C. B. Hergenhan, V. Triolo, and D. M. Ungar. File Machine Startup. August 1, 1978. MF 78-3111-18.

[6] V. Triolo. UNACS Disk Subsystem. June 25, 1979. MF 79-3111-17.

Alan L. Glasser

HO-3168-ALG-alg

Alan L. Glasser