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The information contained herein is for the use of employees of Bell Laboratories and is not for publication (see GEI 13.9-3)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- Implementation of Large Contiguous Files and Asynchronous I/O in UNIX

Date- January 4, 1974

TM- 74-1352-1

Other Keywords- UNIX
Operating Systems
Time-sharing
Large Files

Author(s)	Location and Room	Extension
Lycklama, H.	MH 7C-211	6170

Charging Case- 39394

Filing Case- 39394-11

ABSTRACT

Large contiguous files and asynchronous I/O have been implemented in the UNIX time-sharing system on a Digital Equipment Corporation PDP-11/45 computer. These features were implemented to aid in handling the large volumes of data required for picture processing research in Center 135. It is now possible to transfer large amounts of data (>512 bytes) directly to or from a user's address space.

Pages Text 12 Other 2 Total 14

No. Figures 0 No. Tables 0 No. Refs. 2

Address Label

DISTRIBUTION
(REFER GEI 13.9-3)

MPLETE MEMORANDUM TO

RRESPONDENCE FILES -HO

OFFICIAL FILE COPY
(FORM E-7770) - PLUS
ONE WHITE COPY FOR
EACH ADDITIONAL
FILING CASE
REFERENCED

DATE FILE COPY
(FORM E-1328)

13 REFERENCE COPIES

10 EXD
1353
1355
1357
1358
135 DPH
13 DIR

ER,W O
LDWIN,GARY L
RRANG,J E
(ER,JEAN-DAVID
WORTH,R H
EN,EDWARD G
D,G D
AINARD,RALPH C
)WN,EARL F
HSBAUM,S J
)RIKIS,Z L
DY,JAMES C
N,GEN M
ISTENSEN,C
)GSTON,A M
LDREN,LARRY A
NDON,J H
LER,C CHAPIN
GHTY,DAVID W
ENBERGER,ROBERT L
ENY,S L
LETTE,DEAN
)RDANO,PHILIP P
DON,P L
LCOCK,ROBERT W
NAY,N B
KELL,BARRY G
VIS,JOHN F
ZYNA,E S
)ICE,CHARLES N
FAUVER,W L
BURTZ,R BRUCE
SEN,ARTHUR B
B,J O
ACS,M E
KLAMA,HEINZ
CUSI,M D
ANZANO,JOSEPH F
KTHALER,MISS G E
)NALD,H S
LER,S E
A,KENT V
NTS,FRANK W

COMPLETE MEMORANDUM TO

MULLER,J F
MURPHY,J ARTHUR
NETRAVALI,ARUN N
NINKE,WILLIAM H
NOOLANDI,J
+PATEL,C K N
PEDERSEN,T J
PERUCCA,JAMES R
PRASADA,BIREN
PRIM,ROBERT C
RENTSCHLER,JOHN A
+RITCHIE,DENNIS M
+ROBERTS,CHARLES S
ROBINSON,FRANK
ROSENBERG,ROBERT L
ROWLINSON,D E
RUBINSTEIN,CHARLES B
SCATTAGLIA,JAMES V
SCHMIDT,ROBERT L
SLICHTER,W P
STORZ,FREDRICK G
SWANSON,GEORGE K
TEWKSBURY,S K
THOMPSON,JOHN S
+THOMPSON,K
THURSTON,R N
+TILLOTSON,L C
+VARNEY,ROBERT C
WALSH,K A
WEBBER,H T
WENGER,J L
WHITACRE,W E
WHITTEN,J H
WILD,J CHRISTIAN
WOLONTIS,V MICHAEL
YAMIN,MRS E E
YOUNG,JAMES A
80 NAMES

COVER SHEET ONLY TO

CORRESPONDENCE FILES -HO

5 COPIES
PLUS ONE COPY FOR
EACH ADDITIONAL
FILING SUBJECT

1271
1273
135
3412
4555
4561
5222
8215
8234
COOSDC

ABRAHAM,STUART A
ACKERMAN,A F
AHO,A V
AHRENS,RAINER B
ALBERTS,BARBARA A

COVER SHEET ONLY TO

ALCALAY,DAVID
ALLEN,JAMES R
ALLES,HAROLD G
ANDERSON,
ANDERSON,D Q
ANDERSON,ROBERT V
ANDERSON,WILLIAM A
APIDAS,E J
ARMBRUSTER,MISS M E
ARMSTRONG,ANGUIN
ARNOLD,GEORGE W
ARNOLD,S L
ARTHURS,EDWARD
ATAL,B S
AVERILL,R M JR
AXON,S L
BACKLAR,MISS MICHELE S
BACKMAN,G A
BARTLETT,WADE S
BAUER,ERICH O
BAUGH,C R
BAYER,DOUGLAS L
BAYER,JESSE A
BENDA,M
BENGTSON,A H
BENJAMIN,O CONNELL J
BERGHOLM,MRS SHARON A
BERGLAND,G DAVID
BERNSTEIN,LAWRENCE
BIAZZO,MARTIN R
BIGELOW,C W JR
BIGELOW,J H JR
BILINSKI,D J
BILOWOS,RICHARD M
BIRCHALL,R H
BIRECKI,MRS K M
BIREN,MRS IRMA B
BITTNER,MRS B B
BLINN,JAMES C
BLY,JOSEPH A
BODEN,F J
BODNAR,J J
BOGERT,P JEFFREY
BOHACHEVSKY,I O
BONACHEA,R N
BOWEN,F W
BOWRA,J W
BOWYER,L RAY
BOYCE,W M
BRADY,PAUL T
BRANDT,RICHARD B
BREECE,HARRY T III
BREWER,F A
BRIGGS,P R JR
BRIGHT,N F
BRISSON,R J
BROWN,COLIN W
BROWN,G W
BROWN,W STANLEY
BUCK,I D
BURDETTE,WILLIAM A
BUTZIEN,PAUL E
BYRNE,EDWARD R
CABLE,GORDON G JR
CAMLET,J V JR
CARRAN,J H
CARUSO,A F

COVER SHEET ONLY TO

CASPERS,MRS BARBARA E
CAVINESS,JOHN D
CEMASHKO,FRED
CHAFFEE,N F
CHAMBERS,J M
CHEN,STEPHEN
CHERRY,MS L L
CLAYTON,DANIEL P
CLEARY,ROBERT W
CLIFFORD,ROBERT M
COBEN,ROBERT M
COHEN,HARVEY
COLE,LOUIS M
COLLIER,ROBERT J
CONNERS,R R
CONNOLLY,C V
CONWAY,MISS MAUREEN A
COOK,THOMAS J
COOPER,C ANTHONY
COPELAND,JOHN A
COPP,DAVID H
CORASICK,MISS M J
COULTER,J REGINALD
COURTNEY-PRATT,J S
CRANE,RODERICK P
CRUDUP,G D
CRUME,LARRY L
CRUSE,MISS BARBARA L
DALE,O BRUCE
DAVIDSON,CHARLES L
DAVIS,MISS E D
DAVIS,MISS L P
DAVIS,R L JR
DAYE,C M
DE JAGER,D S
DETRANO,MRS M K
DEUTSCH,DAVID N
DICKERT,L S JR
DICKMAN,B N
DICK,GEORGE W
DIMINO,L A
DIMMICK,JAMES O
DOLAN,MRS MARIE T
DOLOTTA,T A
DRYDEN,JOHN J
DUBIS,M
DUFFY,FRANCIS P
ECKENRODE,MRS JUDY C
EDELE,JAMES S
EITELBACH,DAVID L
ELLIOTT,R J
ELY,T C
ERB,R W
ESSERMAN,ALAN R
ETRA,RICHARD H
FABISCH,MICHAEL P
FELS,ALLEN M
FELTS,W J
FETTE,CHARLES J
FIGLIUZZI,MISS M E
FISCHER,H B
FISCHER,W C
FISHER,MRS DIDINA
FLANAGAN,J L
FLEISCHER,HERBERT I
FORD,GERARD A
FORT,JAMES W

COVER SHEET ONLY TO

FOUGHT,B T
FOUNTOUKIDIS,A
FOWLER,BRUCE R
FOY,J C
FRANKLIN,MISS C M
FRANK,MISS A J
FRANK,RUDOLPH J
FRARY,MISS EILEEN S
FRASER,A G
FREEMAN,K GLENN
FREENY,ANNE E
FRICKE,J A
FROST,H BONNELL
FULTON,ALAN W
FURCZYK,MRS H E
FU,CHEN
GARCIA,R F
GATES,G W
GAWRON,L J
GAY,FRANCIS A
GEBALA,S G E
GELBER,MISS CHERON L
GEPNER,JAMES R
GERARD,ALLAN
GEYLING,F T
GIBB,KENNETH R
GILBERT,MRS HINDA S
GIMPEL,JAMES F
GITHENS,JOHN A
GLASER,W A
GLOVER,MRS S G
GLUCK,F
GOLABEK,MISS R
GOLDSMITH,L D
GOLDSTEIN,A JAY
GOODRICH,LEWIS M
GRAHAM,R L
GRANDLE,J A JR
GRAVES,DONALD A
GREENBAUM,H J
GREENBERG,JACK M
GREENE,MRS DELTA A
GREENHALGH,H WAIN
GREISEN,MISS K E
GRIPP,MISS N
GROSS,ARTHUR G
GUERRIERO,JOSEPH R
GUTT,DONALD J
HAAS,L J
HAGELBARGER,D W
HAGGERTY,JOSEPH P
HALFIN,SHLOMO
HALL,ANDREW D JR
HALL,MILTON S JR
HAMILTON,MISS P A
HANSEN,MRS G J
HANSEN,R J
HARASYMIW,J
HARKNESS,MRS CAROL J
HARPER,MRS G F
HARRISON,NEAL T
HARTMANN,MRS NANCY L
HARTMANN,ROBERT H
HARTMAN,D W
HARTWELL,WALTER T
HARUTA,K
HASZTO,EDWARD D

+ NAMED BY AUTHOR

> CITED AS REFERENCE SOURCE

...
487 TOTAL

RADY,J E; MH 7B201;

TM-74-1352-1 TOTAL PAGES 15

ET A COMPLETE COPY:

E SURE YOUR CORRECT ADDRESS IS GIVEN ON THE OTHER SIDE.

OLD THIS SHEET IN HALF WITH THIS SIDE OUT AND STAPLE.

IRCLE THE ADDRESS AT RIGHT. USE NO ENVELOPE.

PLEASE SEND A COMPLETE COPY TO THE ADDRESS SHOWN ON THE OTHER SIDE.
NO ENVELOPE WILL BE NEEDED IF YOU SIMPLY STAPLE THIS COVER SHEET TO THE COMPLETE COPY.
IF COPIES ARE NO LONGER AVAILABLE PLEASE FORWARD THIS REQUEST TO THE CORRESPONDENCE FILES.

Bell Laboratories

subject: Implementation of Large Contiguous Files and Asynchronous I/O in UNIX - Case 39394

date: January 4, 1974

from: H. Lycklama

TM-74-1352-1

Memorandum for File

INTRODUCTION

The UNIX operating system (ref. 1) running on the DEC PDP-11/45 computer was chosen to support picture processing research in Center 135 (ref. 2) because of the large amount of support software which had been written for it. However picture processing requires the dynamic creation, deletion and accessing of large files (up to 32,000,000 bytes). The version of UNIX available at the time was limited to 65,536 byte files. There was also the need to be able to read and write large amounts of data for real-time applications in an asynchronous manner. These requirements have led to the design and implementation of large contiguous files and asynchronous I/O within the framework of the UNIX operating system as described below.

LARGE CONTIGUOUS FILES

Large contiguous files are implemented within the framework of the UNIX file system. That is, the names of the files follow the UNIX convention of a simple hierarchical structure. Each large contiguous file has an inode associated with it which contains a bit in the "flag" word indicating that it is a large contiguous file and is to be treated specially. The bytes in the inode for such a file are used as follows:

0-1	flags
2	number of links
3	user ID of owner
4-5	size(least sig. word) in bytes
6	volume number of pack on which file exists
7-21	must be zero
22-25	creation time
26-29	modification time
30-31	size(most sig. word) in bytes

The flags are used as follows:

100000	inode is allocated
040000	directory
020000	file has been modified (always on)
010000	large file
004000	big contiguous file
000040	set user ID on execution
000020	executable
000010	read, owner
000004	write, owner
000002	read, non-owner
000001	write, non-owner

These files are implemented only for RP02 and RP03 disk packs. The RP02 disk pack contains 203 cylinders of which only the first 200 are used for the file system. The RP03 disk pack contains 406 cylinders of which only the first 400 are used for the file system. For both packs, each cylinder has 20 tracks of 10 (256 word) sectors each. Thus the total capacity of an RP02 disk pack is 20M bytes whereas that of the RP03 disk pack is 40M

bytes.

In a typical UNIX installation, one pack is used as a permanently mounted system pack with one or more file systems of 7000 sectors each on the pack. Any number of other packs may be mounted or dismounted without disrupting the operation of the UNIX time-sharing system. On the system pack, the area not reserved for the UNIX file systems are set aside for contiguous files. On the other dismountable packs, the complete pack is used for the allocation of contiguous files. For example, for an RP03 system pack with 5 UNIX file systems the last (80000-5*7000) = 45000 sectors on the pack are used for contiguous files. On a dismountable RP03 disk pack, the total 80000 sectors are used for contiguous files.

The contiguous files are allocated on the disk area outside of the file systems defined by the UNIX bit maps. The area on each pack used for contiguous files is defined by a volume label at the beginning of each dismountable pack (rp1 ... rp7) . The volume label for the system pack occurs just beyond the end of the UNIX file systems. Files are allocated in units of a track. In the contiguous file area, tracks are assigned as follows:

```
1          volume label and bit map
2-x        VTOC entries
(x+1)-n    contiguous file areas
```

where (x-1) tracks are devoted to VTOC entries and the last (n-x) tracks are devoted to the actual contents of the contiguous files. Unless otherwise stated, the following values of x, n are used:

	x	n
RP02 system pack	20	500
RP03 system pack	20	4500
RP02 dismountable pack	20	4000
RP03 dismountable pack	20	8000

The sectors in the first track are used as follows:

1	volume label
2	bit map (bit per 2 tracks)
3-10	unused

The words in the first sector of the first track are used as follows:

0	volume number
1-2	date labelled
3	number of tracks available for file system
4	number of tracks for label
5	number of tracks for VTOC entries
6	size of VTOC entry (in bytes)
7-49	character string label supplied by pack owner
50-254	unused (zero)
255	checksum

The next sector contains the bit map for the contiguous file area on the pack. Each bit represents 2 tracks. A 1 bit indicates an un-allocated track whereas a 0 bit indicates an allocated track.

The VTOC entries currently consist of 64 bytes of information on the contiguous file which it describes. Thus one sector contains 8 VTOC entries. The VTOC entries contain sufficient information to reconstruct the bit map if required. The bytes in a VTOC entry are used as follows:

0-31	complete path name of file
32-33	inode of file
34-35	idev of inode
36-39	size of file (in bytes)
40-41	record size (in bytes)
42-43	starting track number
44-45	number of consecutive tracks allocated
46-61	other extents
62-63	checksum

To find the starting disk sector address of a contiguous file the i-node and idev number of the file are hashed to obtain a pointer to a VTOC block on disk. This disk sector is read in and searched for the given i-node and idev, which are the keys to the VTOC entry.

To support these large contiguous files, the following system calls have been added to the UNIX operating system:

```
sys alloc
sys pckm
sys pcku
sys dseek
```

These system routines are invoked by passing the address of the list of arguments in r0.

alloc: - allocate disk space for a contiguous file

```
        mov     $arg,r0
        sys     alloc
        ...
        ...
arg:    name
        mode
        packvol
        ntracks
        rsize
```

where

name - points to null-terminated string naming a file

mode - mode bits of file as in "sys creat"

packvol - pack volume number

ntracks - number of consecutive tracks to be allocated

rsize - record size

pckm: - mount given volume number on given drive number

```
        mov     $arg,r0
```

```
        sys     pckm
        ...
        ...
arg:    drivno
        packvol
```

where

drivno - physical drive number

packvol - pack volume number

pcku: - unmount given volume number

```
        mov     $arg,r0
        sys     pcku
        ...
        ...
arg:    packvol
```

where

packvol - pack volume number

dseek: - move read/write pointer by double word offset

```
        mov     $arg,r0
        sys     dseek
        ...
        ...
arg:    filed
        offset
        ptrname
```

where

filed - file descriptor (0 - 9) refers to file open for reading or writing

offset - double word offset pointer

ptrname - 0 - pointer is set to offset

1 - pointer is set to current location plus offset

2 - pointer is set to size of file plus offset

Normally the system pack is mounted when the UNIX system is

booted up and "/etc/init" is executed. Other packs must be mounted specifically. To enable the user to mount and dismount other packs at will and also to allocate disk space for contiguous files easily, the following commands have been written:

packf

This program allocates space on a given volume number with sufficient consecutive space to hold the given number of records. Space is allocated in 2 track quanta. Packf is invoked by means of:

```
packf name mode volume rsize nrecords
```

where name - pathname of file to be created
mode - mode bits of file
volume - pack volume number
rsize - size of a record
nrecords - maximum number of records in file

packi

This program initializes a disk pack for large contiguous files with all VTOC entries zeroed out and a label put at the beginning of the pack. The bitmap is appropriately initialized. The program may only be invoked by the super-user as follows:

```
packi drive volume label
```

where drive - physical drive number
volume - volume number to be put on pack
label - up to 80 character user specified label

packm

A pack is mounted on the given drive number by means of:

packm drive

where drive is the given drive number. Before a volume can be mounted the label block checksum is verified. Upon a successful mount the volume number of the pack is printed out.

packu

A given volume number is dismounted on a drive by means of:

packu volume

where volume is the given volume number that is to be dismounted.

packl

In order to find out what contiguous files exist on a volume on a given drive, one may invoke:

packl drive

where drive is the given drive number. This program will print out all the pertinent information about the volume mounted on the given drive including a list of all files on the volume. It will essentially give a synopsis of the information contained in the volume label sector and give an indication of how many tracks of the volume have been allocated for files and how many tracks are left unallocated and may still be used. A file map is printed out indicating which VTOC block has been used, where the inode for the file exists, the name of the file, its size in bytes and

the total number of tracks which have been allocated for the file.

Two UNIX commands have been modified to give the user more information about his contiguous files. The "ls" command will now indicate whether or not a file is contiguous by means of a "c" in the mode bits of the file description. The "stat" command will indicate the mode of a file (whether contiguous or not) and will also give the true size of the file (in bytes) making use of the most significant size word in the inode.

As far as UNIX code is concerned, all contiguous files may be accessed in the normal way by means of the standard UNIX system calls such as "open, close, read, write, stat". When a "sys creat" is done on a contiguous file, its length is truncated to zero but the space which had been allocated for it is left intact. To make full use of these large contiguous files, one can utilize the asynchronous I/O features which have been added to UNIX.

ASYNCHRONOUS I/O

For some real-time applications involving large amounts of data, it is often necessary to perform more sophisticated i/o operations than possible with the standard file read and write operations in UNIX. Specifically it is desirable to be able to do I/O directly to or from the user's address space without the use of the system side buffers. The reasons for this are two-fold - reduce system overhead time and allow the input or output of more than 512 bytes of data at a time. In some cases one

would also like to initiate more than one transfer of data simultaneously and then wait for them to finish separately thus reducing transfer set-up times where these are critical. For instance, using asynchronous I/O directly into the user's area, one can transfer a track of data (2560 words) from an RP disk in no more than two revolutions of the disk. Using the standard UNIX read would take more than 10 revolutions of the disk as 10 separate I/O operations are required.

Asynchronous I/O routines have been incorporated into the UNIX operating system to allow one to initiate I/O directly to or from such block-oriented devices as magtape, dectape and all disk devices. For magtape specifically, one is now able to read or write records which are not necessarily 512 bytes long. The implementation of large contiguous files also allows one to initiate asynchronous I/O to or from these files. An error condition will be indicated however if one attempts to initiate asynchronous I/O to or from a standard UNIX file.

Two new system calls have been added to the UNIX operating system in order to implement asynchronous I/O:

```
sys srtio
sys statio
```

srtio:

This call is invoked to initiate asynchronous I/O. Its calling procedure is the same as for "sys read" and "sys write":

```
(file descriptor in r0)
sys     srtio;buffer;nbytes
(system buffer descriptor index in r0)
...
```

The file descriptor is the word returned from a successful open, creat or alloc and determines whether the I/O operation to be initiated is a read or a write. Here "buffer" is the address of the buffer into which or from which the "nbytes" of data are to be transferred. Upon return from the system the I/O has not been completed yet, but the system buffer descriptor which has been allocated for this I/O transfer is returned in r0 and must be remembered when testing the status of this particular I/O transfer. The error bit will be set if the I/O could not be initiated. Conditions for error may include: bad buffer address, "nbytes" which would cause transfers outside the user's address space, bad file descriptor or no more system buffer descriptors available. Currently only 4 buffer descriptors are available in the system to be used for asynchronous I/O transfers. This number is an assembly parameter and may be increased if need be.

statio:

This system call is invoked to check the status of a given asynchronous I/O transfer as follows:

```
        mov     $bufst,r0
        sys     statio
        ...
        ...
bufst:  bptr
        flags
        nbytes

where   bptr - index to the system buffer descriptor which
               describes the I/O transfer requested as passed
               back by "sys srtio"
        flags - status of this particular I/O operation
                    04000 - read active
                    02000 - read outstanding
                    01000 - write active
                    00400 - write outstanding

        nbytes - number of bytes returned.
```

The user must pass the index to the system buffer descriptor which describes the I/O transfer requested as passed back by "sys srtio" in "bufst". The flags and nbytes are returned to him in "bufst+2" and "bufst+4" respectively.

All asynchronous I/O initiated must be waited for, but the order in which they are waited for is not important. The flag bits are returned as zero upon a successful completion of the I/O transfer. The error bit will be set upon an error condition detected such as physical I/O errors or if the user is not the owner of this system buffer descriptor. While there is any outstanding asynchronous I/O for a user, he is guaranteed not to be swapped out. Upon exiting from a process all asynchronous I/O started is first waited for.

A simple example of the use of these asynchronous I/O routines is given in Appendix A. The program copies data from an RP02 drive 0 to an RP02 drive 1, one track at a time. Total execution time is less than 5 minutes compared to almost 40 minutes by using the standard "sys read" and "sys write" routines in UNIX.

H. Lycklama

MH-1352-HL-JER

H. Lycklama

Att.
References
Appendix A

References

(1) MM-71-1273-4,"The UNIX Time-Sharing System"
D. M. Ritchie.

(2) J. D. Beyer, Dept. 1353.

Appendix A

```
/ copy rp0 to rp1 using asynchronous io routines

srtio = 50.
statio = 51.
nsect = 10.

        sys     open;rp0;0
        bes     oerr
        mov     r0,rptr
        sys     open;rp1;1
        bes     oerr
        mov     r0,wptr
        mov     $200.*203.,r3
2:
        mov     rptr,r0
        sys     srtio;buffer;nsect*512.
        bes     rerr
        mov     r0,bufst
1:
        mov     $bufst,r0
        sys     statio
        bes     serr
        mov     $bufst,r0
        bit     $7400,2(r0)
        bne     1b
        mov     wptr,r0
        sys     srtio;buffer;nsect*512.
        bes     werr
        mov     r0,bufst
1:
        mov     $bufst,r0
        sys     statio
        bes     serr
        mov     $bufst,r0
        bit     $7400,2(r0)
        bne     1b
        sub     $nsect,r3
        bne     2b
        sys     exit
oerr:   4
rerr:   4
werr:   4
serr:   4
rptr:   0
wptr:   0
rp0:    </dev/rp0\0>
rp1:    </dev/rp1\0>
        .even
bufst:  .=.+6
buffer: .=.+[nsect*512.]
```