Hullestue

TITLE: CR FILE SYSTEM PSP

DOCUMENT NO: CSS/910/EWP/0001

PREPARED BY: Peter Haumann Peter Haumann

APPROVED BY: Jørgen Høg Jørgen Høg

AUTHORIZED BY: Jørgen Høg Jørgen Høg

DISTRIBUTION: AMC (5)
CR (30)

ISSUE:	1	2						
DATE:	780911	790226						

400-571-2

CR FILE SYSTEM PSP	sign/date JHØ/790226	page
	repl PH/780911	project

PAGE ISSUE RECORD AND CHANGE LOG.

PAGE	ISSUE 1	2	3	4	5	6	7	8
01								
02								
03								
04								
05								
06								
07								
08								
09								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								
21								
22								
23								
24								
25								
26								
27								
28								
29								
30								
31								
32								
33								
34								
35								
36								
37								
38								
39								
40								
41								
42								
43								
44								
45								
46								
47								
48								
49								
50								
51								
52								
53								
54								
55								
56								
57								
58								
59								
60								
61								
62								
63								
64								
65								
66								
67								
68								
69								
70								
71								
72								
73								
74								
75								
76								
77								
78								
79								
80								
81								
82								
83								
84								
85								
86								
87								
88								
89								
90								
91								
92								
93								
94								
95								
96								
97								
98								
99								
100								

ISSUE	DATE	PREPARED BY	APPROVED BY	AUTHORIZED BY
1	780911	PH	[illegible]	[illegible]
2	790226	PH		

CHRISTIAN ROVSING A/S	DOCUMENT CHANGE NOTICE	1. SHEET 1 OF 1 DATE 800124

2. CDC NO.	3. DOCUMENT NO. CSS/910/EWP/0001	4. ISSUE 2	5. DCN NO. 1
6 REL.CAT. ☐ I ☐ II	7. PROJECT. / 8. CONTRACT NO.	9. REF.	

10. DOCUMENT TITLE CR FILE SYSTEM PSP

THE DOCUMENT IDENTIFIED IN BLOCKS 3 AND 4 HAS BEEN CHANGED AS SHOWN IN BLOCK 11. THE CHANGED PAGES AUTHORIZED BY THIS DCN ARE ATTACHED. ALL CHANGES AUTHORIZED THROUGH DCN's CONSTITUTE TOGETHER WITH THE ORIGINAL ISSUE SHOWN IN BLOCK 4 THE CURRENT VERSION OF THE DOCUMENT.

11. DOCUMENT CHANGE

REMOVE PAGES	DATED	INSERT PAGES	DATED
		DCN PAGE (FOLLOWING THE FRONT PAGE)	
		page 38	800124
		page 39	800124
		page 40	800124
		page 41	800124
		page 42	800124
		page 43	800124
		page 44	800124
		page 45	800124
		page 46	800124

13. PREPARED BY Gordon McAlpine	DATE 800124
14. APPROVED BY	DATE
15. AUTHORIZED BY	DATE

HKJ-79-12.

~~400-593/1~~

LIST OF CONTENTS Page

A

Page

1. SCOPE

This document describes the file system program which is a part of the CR file management system.

The description will refer to the program as well as to the internal structure and function of the program. First the structures which are built on external storage media are mentioned. Then the structures which exist internally to the file system program follows. The operations which can be performed on these structures are described.

Different features like protection, initialization, and bootstrap loading, garbage collection, and failure tolerance and error recovery are also described.

2. APPLICABLE DOCUMENTS

None.

3. EXTERNAL DATA STRUCTURES

The data structures which are built on the external storage media (disks etc.) are described. First, structures very close to the actual hardware are depicted and successively higher levels of organization are added on top thereof.

3.1 Volumes

The basic external store used by the file system is called a volume. A volume is an ordered set of sectors which are numbered 0,1,2,etc. A sector is a fixed length sequence of bytes (fig. 3.1).

Such external store need not exist as a piece of hardware. Instead it will normally be simulated by software (by a socalled driver process). Such a software unit is responsible for managing a hardware implemented store and for transforming requests concerning a numbered sector into requests to that store.

The purpose of postulating the existence of a software simulated store is to make the file system insensitive to the pecularities of different external storage media. In this way it should be easy to introduce new storage types under the regime of the file system.

0	1	2	3	4	5
6					
12					
18					23

Fig. 3.1:A volume structured as an ordered sequence of sectors.

3.2 Files

A file can be considered as an ordered sequence of blocks, which are numbered o, 1, etc. A block is a fixed length sequence of bytes.

A file is implemented by mapping its blocks onto the sectors of a volume. There is a one-to-one correspondence between blocks in a file and sectors on a volume. The file system supports two different mechanisms for transforming a block number in a file to a sector number on a volume:

- The blocks of a contiguous file are mapped onto a sequence of contiguous sectors on a volume (fig. 3.2). For a file thus organized the number of the sector which contains a given block in the file can be found by adding the number of the block to the number of the sector which contains the first block of the file. A drawback is that the size of a contiguous file can not be expanded after the initial creation of the file (due to possible conflicts in the allocation of sectors to different files).

- The blocks of a random file are mapped onto sectors which are scattered across a volume. The mapping is based on an index which for each block number in the file contains the number of the corresponding sector on the volume. The index itself is also stored on the volume. Index blocks can be linked together to make the size of the file unlimited (except by the amount of volume space).

A

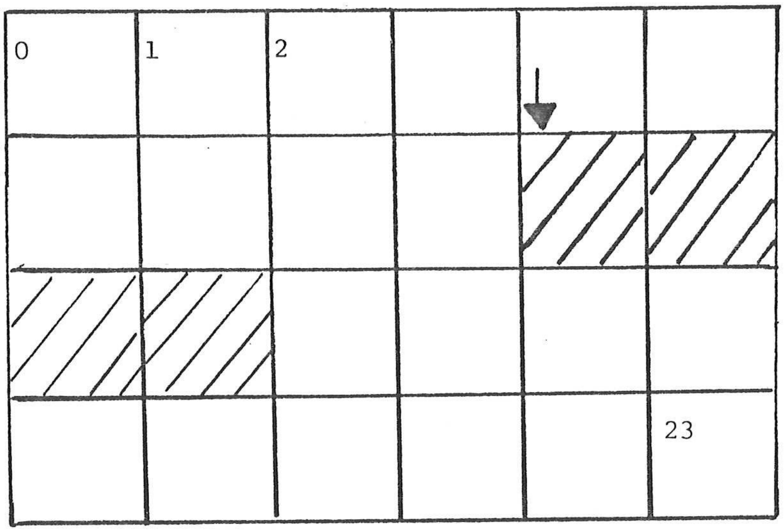

Fig. 3.2: A contiguous file on a volume.

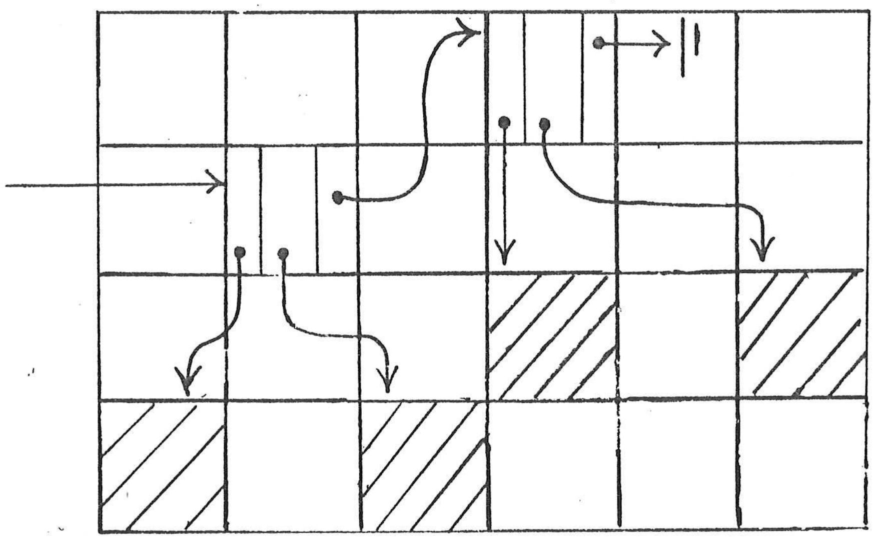

Fig. 3.3: A random file on a volume.
(The file contains 2 index blocks and four data blocks).

3.3 Basic File Directory

A volume can contain several files. Therefore, each volume contains a Basic File Directory (BFD) which acts as a table of contents for the volume (fig. 3.4). Each file on the volume is described by an entry in the BFD. Such an entry contains the information which is necessary to describe the file. Included is information which makes it possible to retrieve the blocks of the file, the size of the file, a list of the users who are authorized to access the file etc.

Since the BFD contains an entry for each file on the volume, a file is uniquely identified by the sequence number of its entry in the BFD. This form of file identification is used in the file system.

To facilitate access to the BFD of a volume, the BFD is also implemented as a file. The BFD should always exist on the volume.

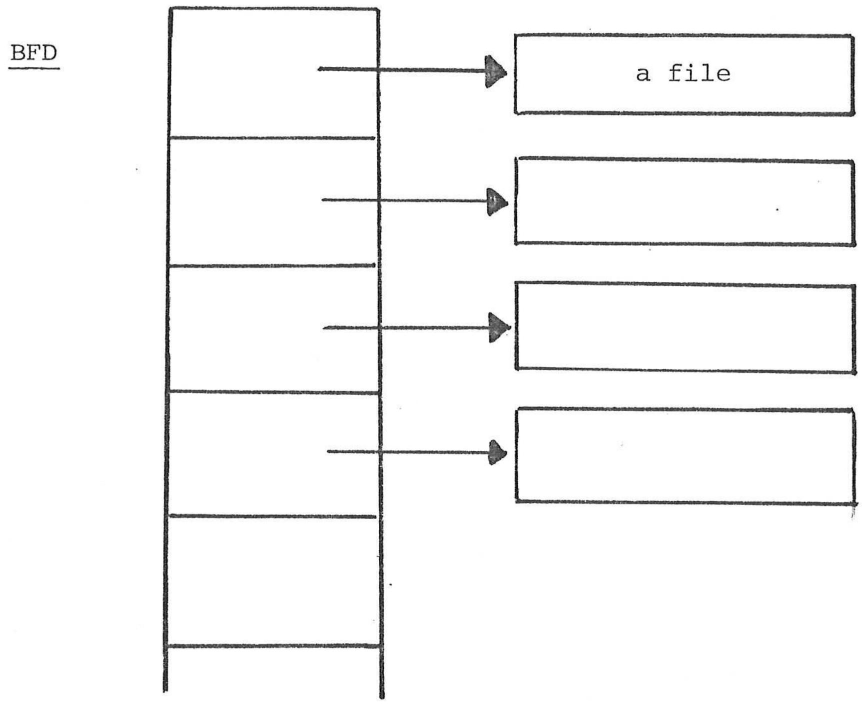

Fig. 3.4: Basic File Directory and four files on a volume. (the sector structure of the volume is abstracted away).

3.4 Symbolic File Directory

Whereas the BFD is concerned with maintaining descriptions of the files on a volume, a Symbolic File Directory (SFD) is concerned with the naming of these files. Naming a file is thus a function which is distinct from describing its attributes.
A SFD functions as a table. Each entry transforms a user defined name into a sequence number of a BFD entry (which is a unique identification of a file). If a SFD is used it is therefore possible to refer to a file by a symbolic name (fig. 3.5).

By implementing a SFD as a file and by allowing several SFD's on a volume, this scheme has been generalized into a multilevel naming structure (fig. 3.6). Since a SFD is itself a file it can now be given a name in another SFD etc. This process can continue to any depth, and thus a hierarchical naming structure for files exists.

Each volume contains a special SFD. This SFD is considered as the root of the naming hierarchy for files. This means that a search for a named file in principle must start in this SFD and then possibly continue through lower level SFD's.
This special SFD should always exist on the volume.

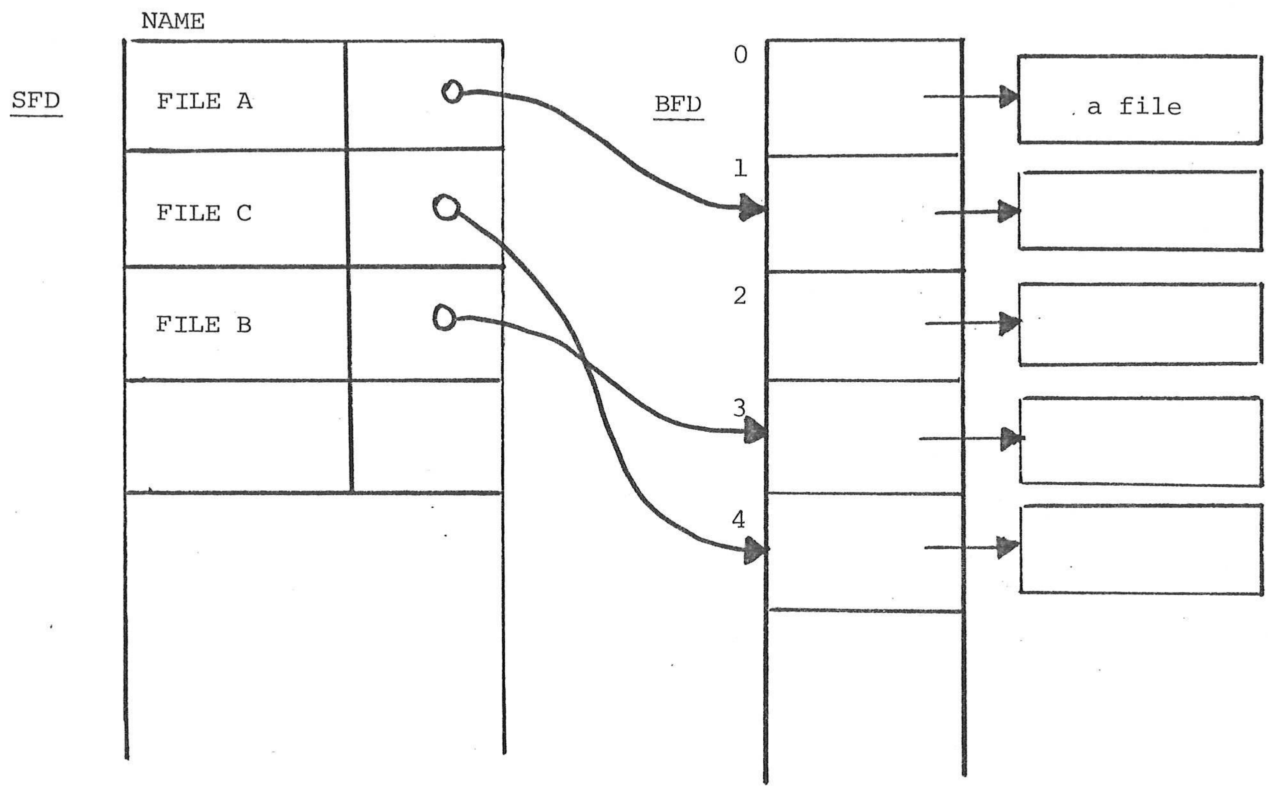

Fig. 3.5: Transformation of symbolic names into file references via a SFD.

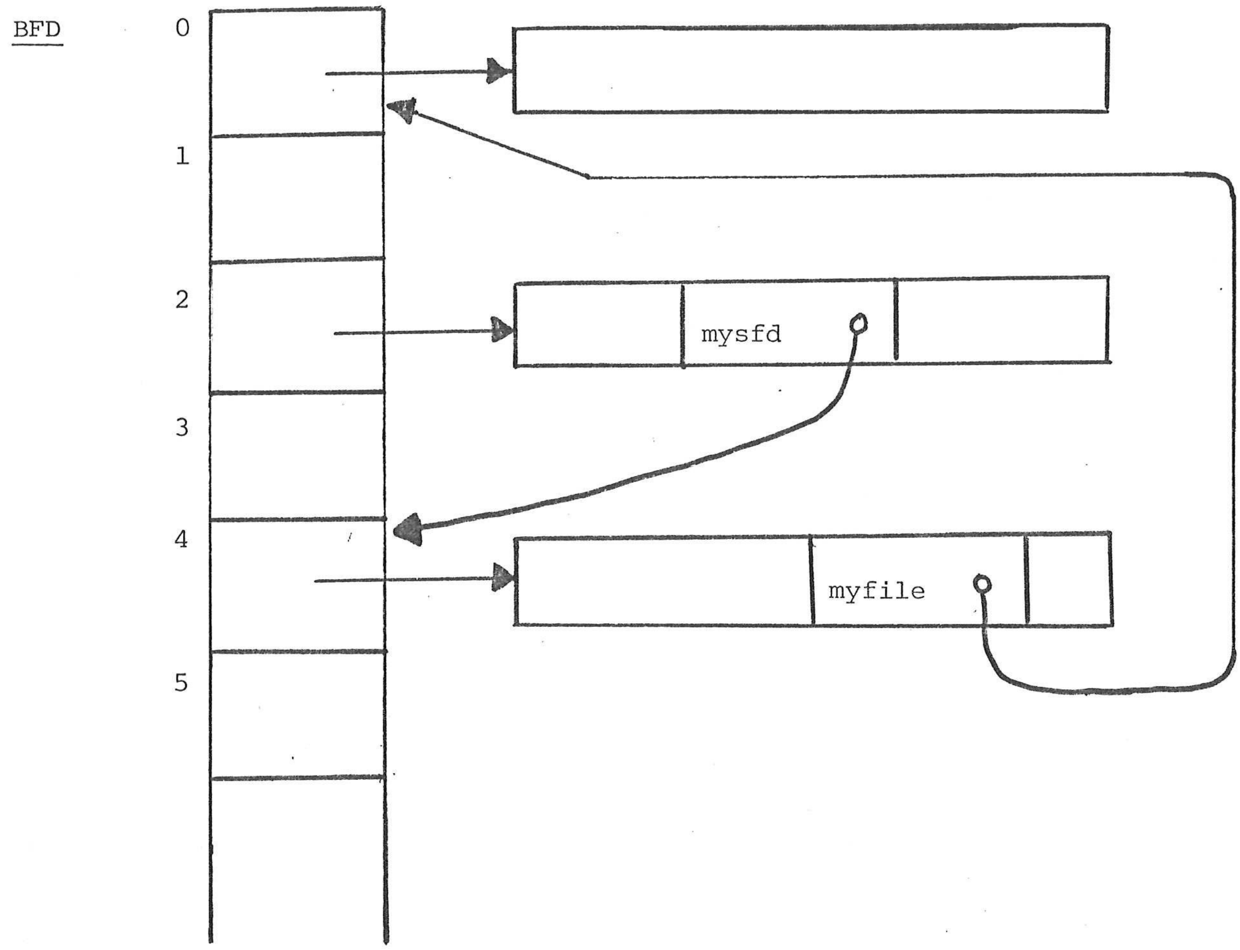

Fig. 3.6: Transformation of symbolic names into a file reference via several levels of SFD. (starting with file #2 in the BFD (which is a SFD) the name 'mysfd' can be transformed into a reference to file #4, which transforms the name 'myfile' into a reference to file #0).

3.5 Homeblock

Each volume contains three special files:

- The Basic File Directory (BFD) which contains a description of the files on the volume.
- The Bit Map which contains information on the allocation status of each sector on the volume.
- The Root Symbolic File Directory which in principle is the starting point for a search of a named file.

These files contain the information which makes it possible to access the rest of the files on the volume. Therefore they should always exist on the volume. Access to these files can be gained through the Home Block (HB) of the volume. Apart from the name of the volume the HB contains the sector address of the description of the BFD (which is actually contained in the BFD).
The HB is the only information on the volume which is not part of a file. Since the HB is always stored on a known address on the volume it can be used to bootstrap the entire file structure.

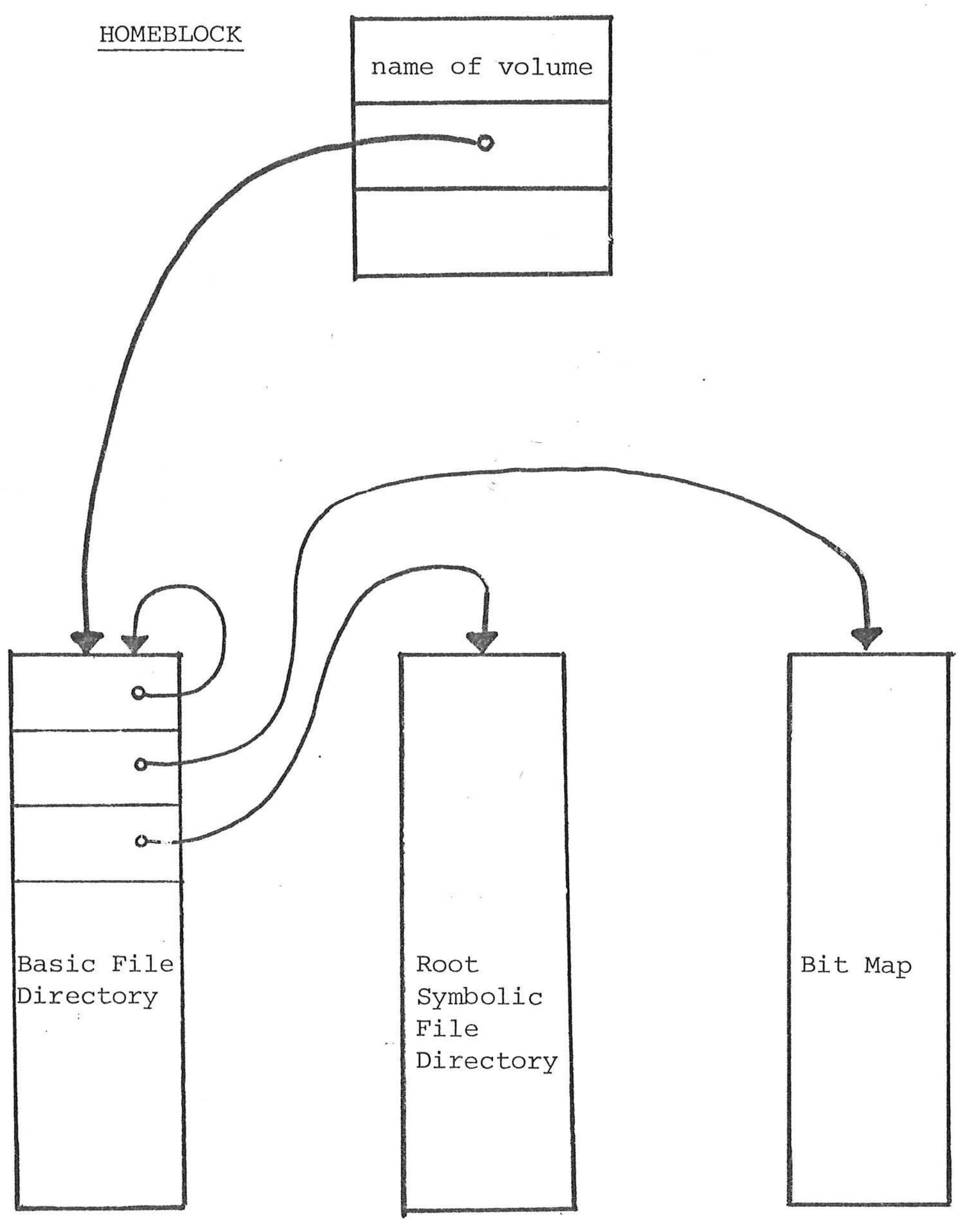

Fig. 3.7: The Home Block and the special files on a volume.

4. INTERNAL DATA STRUCTURES

The internal data structures which are used by the file system are called control blocks. Each control block describes an object which exists externally to the file system.

- User Control Block (UCB)
- Device Control Block (DCB)
- Volume Control Block (VCB)
- File Control Block (FCB)

Fig. 4.1 depicts the interrelationships between control blocks. The different control block types are now described.

4.1 User Control Block

A User Control Block (UCB) represents an active user. When a user wants to use the file system a corresponding UCB must therefore be created and it must be destroyed again when the user finishes his work on the file system. Among the information contained in a UCB is the name of the user and references to File Control Blocks for files which the user has opened. References to a UCB can be made through a symbolic user name.

4.2 Device Control Block

A Device Control Block (DCB) represents a device (disk controller etc.) which can be used by the file system. A DCB contains the information which makes it possible for the file system to use the corresponding device. Examples of such information are the

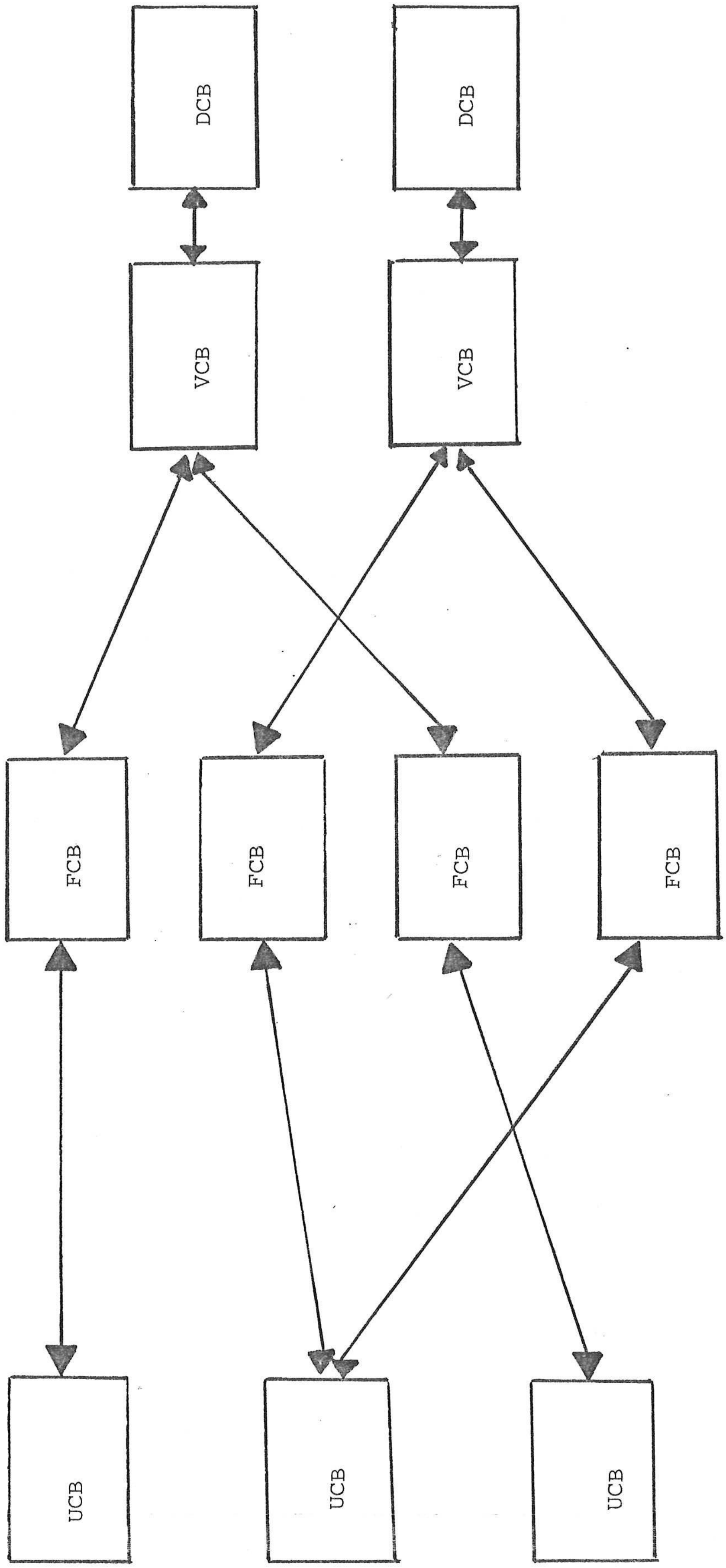

Fig. 4.1: Internal data structures.

name of the device driver process and the device type. References to a DCB can be made through a symbolic device name.

4.3 Volume Control Block

A volume Control Block (VCB) represents a volume which has been mounted on a device. A VCB contains information about the current state of the volume. Among other things a VCB contains references to the FCB's for all open files on the volume. References to a VCB can be made through a symbolic volume name.

4.4 File Control Block

A File Control Block (FCB) represents a file on a volume. There does not exist a FCB for every file which is stored on the volumes. Only files which have been "opened" by some suitable command from a user are represented by an FCB. An FCB contains information which makes it possible to access the corresponding file. Such information is the address of the file on the volume, its size etc. An FCB also contains references to the UCB's for all users who operates on the file and a reference to the VCB for the volume on which the file resides. An FCB is identified by a file descriptor (which is just an integer which can be interpreted by the file system).

5. COMMANDS

The operations which can be performed on the external and internal data structures are now described. The description reflects the set of commands which the user can give to the file system. The input and output parameters for the commands are specified and so is their effect on the external and internal data structures.

5.1 User Commands

The USERON command allows a user to identify another user to the file system and the USEROFF command allows him to make the user unknown again. These commands are a prerequisite for the protection system since each command to the file system must be executed on behalf of a user who is "on".

Command: USERON

Inputs: User name for caller.
User name for another user.

Internal effects: Creates a new UCB. The UCB can be referred to by the user name.

Command: USEROFF

Inputs: User name for caller.
User name for another user.

Internal effects: Destroys the UCB which is identified by the user name.

5.2 Device Commands

The device commands allow specification of the set of devices which can be used by the file system. A device can be included by the ASSIGN command and exclusion of a device is specified by the DEASSIGN command.

Command: ASSIGN.

Inputs: User name for caller
A device name
A device description:
- device type.
- CR80 device address.
- number of sectors on device.

Internal effects: Creates a new DCB which contains a description of the device. This DCB (and the device) can now be identified by the device name.

Other effects: If necessary a driver process which represents the device is created.

Command: DEASSIGN

Inputs: User name for caller
A device name.

Internal effects: Destroys the DCB which is identified by the device name.

Other effects: If necessary the driver process which represents the device is destroyed.

5.3 Volume Commands

The volume commands are used for specification of the set of volumes which can be accessed by the file system. The MOUNT command connects a volume to a device while the DISMOUNT command destroys such a connection.

Command: MOUNT

Inputs: User name for caller
Device name

Internal effects: Creates a VCB which represents the volume and FCB's for the system files on the volume (Basic File Directory, Bit Map, Root Symbolic File Directory). Hereafter the volume is identified by the volume name which is stored in the HOME BLOCK of the volume.

Command: DISMOUNT
Inputs: User name for caller
Volume name

Internal effects: Destroys the BCB which represents the volume and the FCB's which represent the system files on the volume.

5.4 File Commands

The file commands allow users to manipulate files. These commands generally identify a file by a file descriptor (which is just a reference to the FCB for the file). By invoking a CREATE command the user can have a new file created. The DISMANTLE command makes it possible to have the file removed again or at least made inaccessible to the user. By invoking the RESET command all the information in a file can be discarded and the file made "empty". The commands OFFER and ACCEPT allow file descriptors to be transferred between users. A user who is currently using a file can "offer" it to another user, who then must "accept" it to effect the transfer. Finally the protection state of a file (the specification of the users who may access the file) can be changed by the PROTECT command.

Command: CREATE

Input: User name for caller
Attributes of a file:
- volume name
- organization (random or contiguous)
- size

Outputs: A new file descriptor.

External effects: Creates a new file on the specified volume. This entails allocation of necessary storage and insertion of a description of the file into the BFD of the volume.

Internal effects: A new FCB representing the newly created file is created. The user can now refer to the file through a file descriptor.

Command: DISMANTLE

Inputs: User name for caller
A file descriptor

External effects: If the file can not be accessed by any user (apart from the caller) all storage allocated to it on the volume is released. Also its entry in the BFD is cleared.

Internal effects: Breaks the connection between the UCB for the caller and the FCB representing the file. The FCB is destroyed if no other UCB's are connected to it.

Command: RESET

Inputs: User name for caller
A file descriptor

External effects: All storage allocated to the file is deallocated.

Command: OFFER

Inputs: User name for caller
User name for receiver
A file descriptor

Internal effects: Creates a connection between the UCB for the receiver and the FCB. The connection is marked as not yet "accepted".

Command: ACCEPT

Inputs: User name for caller

Outputs: A file descriptor

Internal effects: Marks the connection as "accepted".

Command: PROTECT

Inputs: User name for caller
User name for subject
File descriptor
Access rights

External effects: Specifies the access rights of the subject vis-a-vis a file.

5.5 Naming Commands

The naming commands allows users to minipulate files identified by symbolic names. The basic construct used by these commands is a Symbolic File Directory (SFD) which maps names into unique identifications of files. The ENTER command inserts a reference to a file into a SFD under a user specified name. The LOOKUP command makes it possible to retrieve the file reference again by giving the symbolic name.
The DESCENT command also retrieves a file reference on the basis of a symbolic name.
However, at the same time the directory is DISMANTLED.
The RENAME command changes the name part of an SFD entry. The REMOVE command deletes a SFD entry and it is also possible that the file referred to is removed from the volume (if the file becomes totally inaccessible). Finally, the GETROOT command is used to get hold of the SFD which is at the top of the naming hierarchy of a volume.

Command: ENTER

Inputs: User name for caller
A file name
A file descriptor for a SFD
A file descriptor

External effedts: Creates a new entry in an SFD which maps a file name into a reference to a file.

Command: LOOKUP
Inputs: User name for caller
A file name
A file descriptor for a SFD

Outputs: A file descriptor

External effects: Locates the entry in the SFD which contains the file name

Internal effects: A FCB representing the file which has been located in the SFD is created. This FCB can now be referred to through a file descriptor.

Command: DESCENT

Inputs: User name for caller
A file name
A file descriptor for a SFD

Outputs: A file descriptor

Effects: The same as LOOKUP. However, an implicit call on DISMANTLE is made on the SFD.

Command: RENAME

Inputs: User name for caller
The old file name
The new file name
A file descriptor for a SFD

External effects: Located the entry in the SFD which contains the old file name. The new file name is inserted instead.

Command: REMOVE

Inputs: User name for caller
A file name
A file descriptor for a SFD

External effects: Clears the entry in the SFD which contains the file name. If the file then becomes inaccessible to all users of the file system, all storage allocated to it is deallocated and its entry in the BFD is cleared.

Command: FETROOT

Inputs: User name for caller
Volume name

Outputs: A file descriptor for the Root Symbolic File Directory of a Volume.

5.6 Transfer Commands

The transfer commands are used to bring about the actual transfer of data between the external storage media and the users data buffers.
Due to the organization of the file system the users data buffers can not be accessed directly. Instead they are accessed via a port number (as implemented by a DMA link). The users data buffers are therefore represented by port numbers.
One set of transfer commands considers the external storage media as volumes made up of sectors. The commands READ SECTORS and WRITE SECTORS transfer information between the user's data area and a contiguous sequence of sectors on the volume. The command FORMAT allows a volume to be formatted and the sectors to be given a specified initial contents.
Another set of transfer commands is used to transfer information between files and the users data area. In this context a file is considered to consist of a sequence of bytes. The bytes are numbered 0,1,2, etc. Any subsequence of the file can be transferred by specifying the first byte in the subsequence and its size.

Command: READSECTORS

Inputs: User name for caller
A device name
Starting sector number
Sector count
A port number

Outputs: Number of sectors
actually transferred.

External effects: Reads a sequence of sectors from the volume and transmits it via a port.

Command: WRITESECTORS

Inputs: User name for caller
A device name
Starting sector number
Sector count
A port number

Outputs: Number of sectors actually transferred.

External effects: Writes into a sequence of sectors on the volume.

Command: FORMAT

Inputs: User name for caller
A device name
Starting sector number
A port number.

Outputs: Number of sectors formatted.

External effects: Formats the volume on a specified device. The formatting starts at a specified sector and continuous until a bad sector is met or until the volume is totally formatted.

Command: MODIFYBYTES

Inputs: User name for caller
File descriptor for a destination file
Start and size of a byte sequence in the file.
A port number.

Outputs: Number of bytes actually transferred.

External effects: Overwrites a byte sequence in the destination file with data received via a port.

Command: APPENDBYTES

Inputs: User name for caller
File descriptor for a destination file
Size of a byte sequence in the file
A port number.

Outputs: Number of bytes actually transferred.

External effects: Appends data received via a port to the destination file.

Command: READBYTES

Inputs: User name for caller
File descriptor for a source file.
Start and size of a byte sequence in the file.
A port number.

Outputs: Number of bytes actually transferred.

External effects: Reads a byte sequence from the source file and transmits it via a port.

6. PROTECTION

From the point of view of protection there exists two essentially different classes of objects in the file system. One class of objects for which a protection scheme exists is devices and volumes. Not everybody should be allowed to specify the configuration of peripherals that can be accessed from the file system. The other class is the files on the volumes. The creator of a file must specify the set of users which are allowed to access the file. The protection mechanisms for these classes of objects are now described.

6.1 Protection of Devices and Volumes

Since the commands which change the state of devices and volumes have radical effects and since they are only executed seldomly, a rather crude and simple mechanism is used to protect these objects. A special user is considered to represent the "system" and only this user is allowed to execute these commands.

6.2 Protection of Files

The protection scheme for files assumes that it is possible to specify the access rights of each individual user to each individual file. By access rights is meant the set of operations which a user may perform on a file. Whenever a user performs an operation on a file it is verifyed that the operation is within his access rights to the file. If it is not the operation will not be performed.

6.2.1 Description of the Protection State

To each file on a volume there is connected an Access Control List (ACL). The ACL for a file describes the access rights of each user who is authorized to use the file. When a file is initially created the creator is given the right to access the file any way he might choose. By invoking the PROTECT command the list of authorized users can be changed.

6.2.2 Access Verification

Each time a file is accessed by a user it must be verified that the user has the right to do so. It is of course possible to find the access rights of the user in the ACL for the file. This would however, necessitate accessing the volume. Instead there exist internal data structures - called capabilities - which show the access rights of users to files which have been " opened ". Therefore, the access to files can be controlled without accessing external storage.

7. UTILITY PROGRAMS

There exists a good number of file system operations which advantageously can be implemented as normal CR80 utility programs rather than be implemented directly by the file system.
As motivations herefore can be mentioned:

- Operations requiring much code, and not executed too often should not be part of the resident file system. Instead they should only be loaded and executed as needed.

- Operations requiring an extensive dialogue with the user should not be part of the resident file system. The reason herefore is that it is difficult to achieve communication between a terminal and the file system.

- Operation which should be extensible, ought not be buried in the resident part of the file system.

Moving operations from the resident part of the file system and into utility programs does require facilities such as:

- Loading of the utility programs (possibly with only limited support from the file system).

- Reading/writing/formatting of specified sectors on specified volumes.

The basic utility programs which are necessary for the file system is now described.

7.1 Volume Initialization Program

This program is used to preprocess a volume such that it can be used as file storage by the file system. This entails the following:

- Formatting the volume.
- Handling of bad sectors on the volume.
- Creation of the initial data structure on the volume, which is expected by the file system.
- Insertion of a system bootstrap file on the volume.

7.2 Volume Salvation Program

This program is concerned with the existence and the restoration of the data structures on a volume. It should for instance be executed after a file system crash. The following facilities should exist:

- Verification that the data structure on a volume is as expected by the file system.

- Repair of an improper structure on a volume. This should be done with as little distortion of information as possible.

- Output of statistics on the utilization of the volume.

- Reclamation of unused storage.

7.3 Directory Program

This program allows the user to inspect and handle the directory structure on a volume.

7.4 Backup and Archival Program

This program allows files to be transferred to auxiliary storage devices for backup and archival purposes. Mechanisms also exist, which allows resotration of files to volumes.

8. INITIALIZATION

T.B.S.

APPENDIX A

EXPLANATION OF FMS COMPLETION CODES

This section explains the meaning of the completion codes which can be returned by the File Management System.

If a FMS command is completed without any errors, then the completion code IO OK (zero) is returned.

System Errors

#300 DCB POOL EMPTY:
#301 UCB POOL EMPTY:
#302 FCB POOL EMPTY:
#303 CAP POOL EMPTY:

These errors mean that an internal FMS data structure is full and therefore the required command could not be completed. Such errors do not destroy the integrity of the FMS nor the volumes that it is handling.

#30D DRIVER TABLE FULL: This means that no drivers are free to be assigned to a new disk. Again, it does not cause the FMS to crash or become inconsistent.

OTHERS All other system errors mean either that an inconsistent volume has been mounted or that there is a programming error within the FMS. If the former is the cause, then (1) all other volumes should be demounted, (2) the User and File Processors should be shut down and then bootstrap loaded and (3) the Disk Salvation utility program should be run on the inconsistent volume using consistent System and Work Volumes. Otherwise, step (2) should be taken and the System Group of the Computer Systems Department should be informed of the error.

User Errors

400 NON-EXISTING DEVICE: This means that a FMS command was requested with a "device name" as a parameter, and that this device name was unknown to the FMS; that is, no device with that name was assigned.

401 ILLEGAL DEVICE KIND: This means that an Assign command was requested with a "device kind" parameter which was unknown to the FMS.

402 ILLEGAL CR80 ADDRESS: This means that an Assign command was requested with a "CR80 address" parameter which did not represent a valid hardware address.

403 DEVICE NAME IN USE: This means that an Assign command was requested with a "device name" parameter which was the same as an already assigned device.

404 ILLEGAL UNIT: This means that an Assign command was requested with a "unit" parameter which was not within the range of unit numbers that can be handled by the disk driver for the given device.

405 ILLEGAL SUBUNIT: Similar meaning to ILLEGAL UNIT.

406 WRONG VOLUME NAME: This means that a Mount command was requested with a "volume name" parameter different from the volume name which was present in the Home Block of the given device.

407 NON-EXISTING VOLUME: This means that a FMS command was requested with a "volume name" parameter which was not the same as any mounted volume.

408 VOLUME MOUNTED: This means that an FMS command was requested on a device (specified by a "device name"), Which already had a volume mounted on it. The Deassign, Mount, Read and Write Sector, and Format commands are not allowed on a mounted device.

409 DIFFERENT VOLUMES: This means that an Enter command was requested with "subject" and "directory" parameters which specified files that lay on different volumes.

40A ILLEGAL FILE DESCRIPTOR: This means that a FMS command was requested with a "file descriptor" which did not identify an open file.

40B ILLEGAL FILE ORGANISATION: This means that a Create command was requested with an "organisation" parameter other than Contiguous, Random or Directory.

40C ILLEGAL ALLOCATION SIZE: This means that a Create command was requested, for a Contiguous file, with an "allocation size" parameter less than zero.

#40D ILLEGAL AREA SIZE: This means that a Create command was requested, for a Random or Directory file, with an "area size" parameter less than zero.

#40E ILLEGAL WRITE: This means that a Reset, Append bytes or Modify bytes command was requested with a "file descriptor" which specified a Directory file.

#40F Not used at present.

#410 FILES OPEN: This means that a Dismount command was requested for a volume which had files open on it.

#411 NO FILE TO ACCEPT: This means that an Accept command was requested by a user (process) which had not been offered any files.

#412 NON-EXISTING USER: This means that a FMS command was requested with a "user name" which did not match a known user; that is which did not correspond to a user name given to a Useron command.

#413 USER ALREADY ACTIVE: This means that a Useron command was requested with a "user name" parameter identical to that of an already active user.

#414 NO CONNECTION: This means that a FMS command was requested, by a user (process), with a "file descriptor" parameter representing a file which was not open to that user.

A

#415 Not used at present.

#416 ILLEGAL CALLER: This means that a FMS command which can only be executed by the System User was requested by a User other than the System User.

#417 OTHER USERS: This means that a Reset command was requested on a file which was open to more than one user.

#418 Not used at present.

#419 OUT OF RANGE: This means that a FMS command to read/write sectors or read/modify bytes was requested with an illegal parameter. For the sector handling commands (including Format), the "sector count" parameter is illegal if it is less than zero and the "first sector" parameter is illegal if it is less than zero or if first sector + sector count is greater than the number of sectors on the device. For the Read bytes command, the "first byte" parameter is illegal if it is less than zero. For the Modify bytes command, the "first byte" parameter is illegal if it is less than zero or greater than the file's size.

# 41A	DISK DRIVER FAILURE:* This means that the File System requested a disk driver to perform an operation and that the result was not successful. This can be for many reasons. For example, the user may have requested an Assign command for a device with the same CR80 address, unit number and subunit number as an already assigned device. Another example is the Assign command with a CR80 address parameter representing an unused hardware address. Another example is an attempt to read/write from/to a disk that is switched off. Another example is an attempt to write to a write-protected disk. Finally, this completion code is also used to indicate a hardware failure in a disk unit.
# 41B	FILE FULL: This means that a Modify or Append bytes command was requested which would have resulted in a file being greater than its maximum possible size.
# 41C	ACCESS CONTROL LIST FULL: This means that an Enter or Protect command was requested, which required new access rights to be entered in a file's BFD entry and that there was no more space in the Access Control List of this entry to hold this access control information.
# 41D	PROTECTION FAILURE: This means that a user requested a FMS command on a file to which he was not allowed the access associated with the command.

* This completion code may be expanded into several new ones in the future.

#41E	Not used at present.
#41F	Not used at present.
#420	ILLEGAL DIRECTORY: This means that an Enter, Lookup or Descent command was requested with a "directory file descriptor" parameter which did not identify a directory file.
#421	NAME ALREADY EXISTS: This means that an Enter or Rename command was requested with a "name" parameter identical to the name of a file already present in the directory specified by the "directory file descriptor" parameter.
#422	NON-EXISTING NAME: This means that a Lookup, Descent, Rename or Remove command was requested with a "file name" parameter which did not match the name of any file in the directory specified by the "directory file descriptor".
#423	NOT ALLOCATABLE: This means that a FMS command was requested which required more space to be allocated to a file and that there was not enough free space left on the volume to fulfil this demand.
#424	VOLUME THRESHOLD EXCEEDED: This means that a FMS command was requested which required more space to be allocated to a file and that this would have meant exceeding the Volume's Threshold.

\# 425 FILE THRESHOLD EXCEEDED: This means that a FMS command was requested which required more space to be allocated to a file and that this would have meant exceeding the file's Threshold.

\# 426 VOLUME THRESHOLD TOO LARGE: This means that a Set Volume Threshold command was requested with a "volume threshold" parameter which was greater than the number of sectors on the device.

\# 427 Not used at present.

\# 428 ILLEGAL FILE INFORMATION TYPE: This means that a Get File Information command was requested with a "information type" parameter that was not one of the valid information types.

DMA Errors

# 501	PORT ERROR:	The meaning of these can be found in the DMA Driver specification. They can only occur if there is a programming error in the File System.
# 502	ILLEGAL DMA COMMAND:	
# 505	ABORTED:	

503 TRANSFER LIST ERROR: This means either that there is an error in the File or I/O System, or that a user has tried to communicate directly with the File System and has not conformed correctly to the communication protocol involved in this.

505 TRANSMISSION ERROR: This means that there was a DMA transmission failure, during a data transfer. Amongst the possible causes are a parity error in one of the processors or a failure in the DMA hardware.