CFSDSG.MEM

CFSDSG.MEM;1

CFSDSG.MEM

LSCS FORM=FLAG

TIME QUEUED			TIME PRINTED	
17-OCT-85 16:01			17-OCT-85 16:05	
LAYOUT	**OVERLAY**	**PAPER TYPE**	**FILE TYPE**	**COPIES**
P75	NONE	3_HOLE	DEFAULT	0001

1.0 Introduction

CFS-20 operates at the level of the TOPS-20 file system. This is accomplished by "distributing" the file system over the CFS "network". This distribution implies several things:

1. Processors must agree on file operations previously of concern only to a single processor.

2. The existing file system data base had to be extended to accommodate additional information.

3. Other, peripherally related, parts of TOPS-20 were affected, e.g. job numbers.

This document will describe the specifics of the CFS kernel, CFSSRV, as well as the extensions and modifications of the file system. Since this is a design document, it is assumed the reader is familiar with the functional specification for CFS-20.

2.0 CFSSRV

The module CFSSRV contains the kernel, or low-level, routines that implement CFS-20. The important pieces of CFSSRV are:

1. The resource manager/creator

2. The "voter"

3. The SCA connection manager

4. Initialization

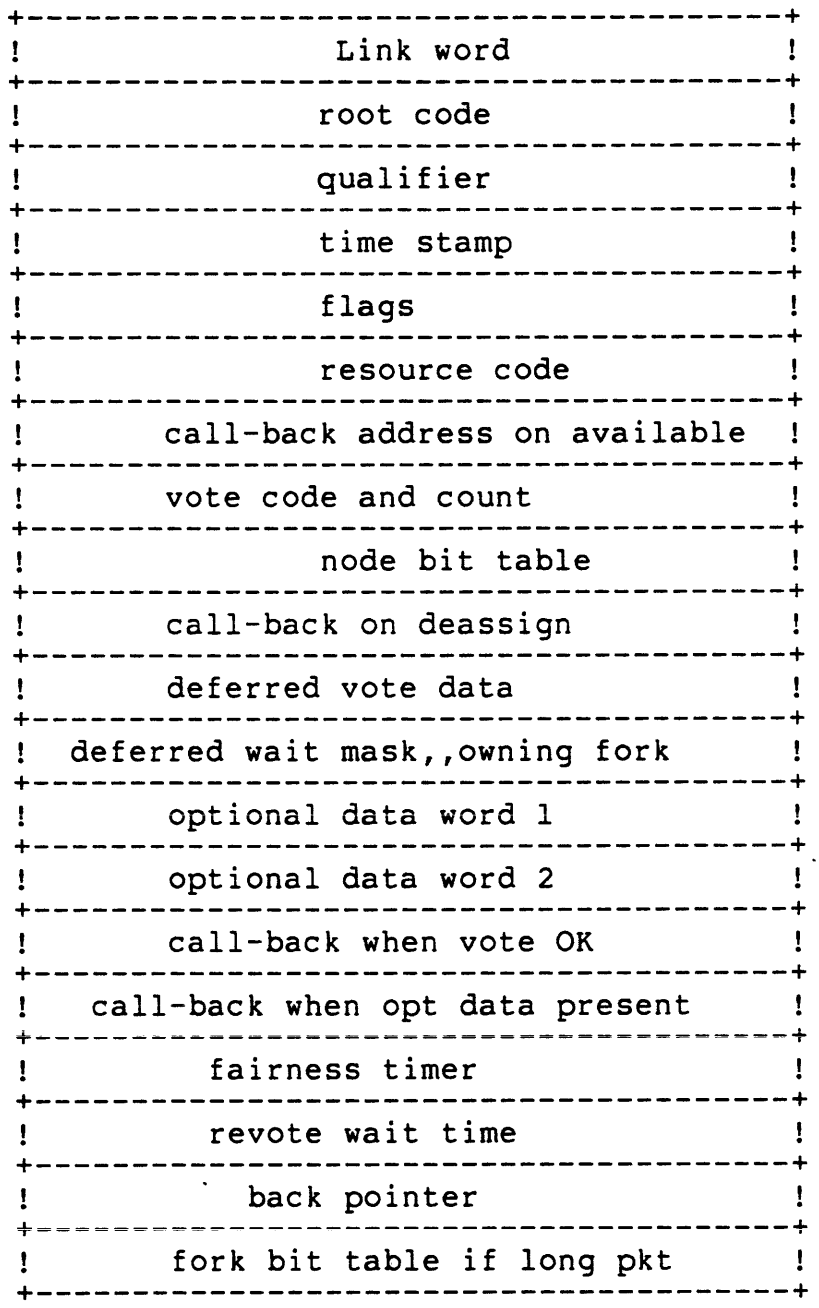
In addition, it contains a number of "interface" routines used by other system modules and services for interfacing to the kernel routines.

2.1 Resource manager

CFS resources are kept in a hash table managed by CFSSRV. The hash table contains "hash chains" built as collisions occur. Each entry represents some resource, but the manager has no inherent knowledge of the actual resource represented. Each resource block contains identifying information, state information, and a number of coroutine addresses that are used when certain events occur.

There are two types of resource blocks: long blocks and short blocks. A long block is the same as a short block except that it contains a bit mask representing the forks in the system. Resources that wish to keep track of blocked forks, and wish to awake the forks when the resource is

available, must use a long block. A block has the following format:



Each resource has a name that is used to compute its hash address and to identify the resource. The name is seventy-two bits and is contained in the second and third words of the hash packet. In addition, a resource may have a resource code that is used for various purpose including:

1. Matching during voting

2. As a mask during garbage collection and searching.

The first word of the name, the root code, is normally a structure name. It is meant to represent a resource class, such as files. The second word of the name, the qualifier, distinguishes specific resources within the class defined by the root name. Therefore, each type of file resource has a unique qualifier that may be constructed based on the type of the resource. For example, the file open token, representing the open state of the file, is simply the file's index block address. The pair of names defines a unique and unambiguous resource.

The various call-back routine addresses must be supplied by the resource requestor. The resource manager has no a priori information about the actions to be taken for each resource. Therefore, when the requestor builds a prototype resource packet, it must fill in any coroutine addresses that it wishes honored. In general, the code that builds the prototype resource packet is located in CFSSRV and therefore is closely allied with the resource manager itself. Call-back addresses need not be provided. In fact, there are resources, e.g. directory locks, that provide no call-back coroutines but rely entirely on the default mechanisms of the manager and the voter.

The resource manager has several entry points. The main entry is CFSGET, but other entries are used if the resource is known to be local (exclusive to this processor).

The manager is responsible for entering packets in the hash table. It is the only agent that places new entries in the hash table.

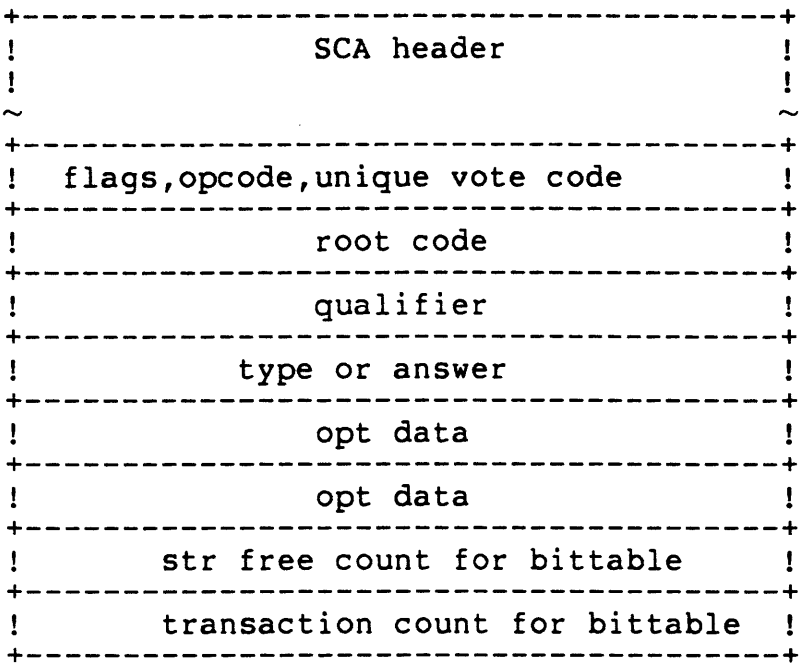
The flags word contains an "access type" that is used to record the level of access of the resource. The manager and the vote processor use this field to determine if an entry or a vote request may be honored.

The manager may delete an old, unused entry, if doing so removes a resource conflict.

2.2 The voter

The voter comes in two parts: the agent that starts votes on behalf of the resource manager, and the agent that processes in-coming vote requests and makes replies. The voter is called when a resource is created or whenever its state changes. The voter is called VOTEW.

A vote request is one of messages that CFSSRV sends to other CFS systems. The general format of a CFS message is:



Requests, or votes, use the opcode .CFVOT. Vote replies use the opcode .CFREP. Replies may contain optional data if the resource is defined as supporting optional data. Examples of optional data are:

1. EOF values for file resources

2. directory allocation for directory resources

In general, the two opt data words contain a value in one of the words and a "transaction count" in the other. The transaction count is used to "age" the data so that the voter may determine which of the reported values is most current. It should be noted that in a large CFS configuration (more than two nodes), a voter may receive different values for the optional data based on the history of the resource. Optional data processing is done through a call-back address, and if that address is not provided in the packet, any received optional data will be ignored.

The voter sends one vote request to each connected host and waits for the replies. The interrupt level part of the voter uses the vote count word in the resource packet to record replies and the process level part of the voter examines this field to determine when the vote has been completed. Also, the flags word contains a flag (HSHYES) that is used to record a "no" reply to the vote.

If the CFS configuration changes during the vote, a "vote restart" flag will be set in the resource block directing the voter to restart the vote. This is done in lieu of

keeping state information about the vote. The restarted vote implicitly cancels the other vote.

The vote processor, the companion routine to the voter, uses the state information in the resource packet to detemine how to reply. A reply may be:

1. Unconditional yes. This may result in a resource being released.

2. Unconditional no.

3. Timed no. This reply has an accompanying retry time that is stored in the hash packet.

4. Conditional or delayed yes.

The vote processor may use one or more of the call-back routines in the resource packet while deciding on the reply. For example, the decision to reply "delayed yes" is made by the call-back routine when the vote is to be OKed. The coroutine may direct the vote processor to change the reply from "unconditional yes" to "delayed yes". This is typically the case for the file access token resource.

2.3 SCA connection manager

The connection manager is repsonsible for maintaining the CFS "host tables" and for interfacing to SCA. Its duties are:

1. Connect to all extant host at start-up

2. Connect to newly on-line hosts

3. Closing duplicate connections

4. sending messages and queuing messages upon credit failures

5. receiving messages.

6. Resending on credit available.

The connection manager maintains four tables that describe connections to other CFS hosts:

1. CFSHST contains the SCA connect ID

2. CFSHNM contains the processor serial number

3. CFHSTS contains the state of the connection

4. CFSNAM contains the DECnet node name

While a connection is being opened, or while a listener exists, the following interpretations apply:

1. If CFSHST = -1, this is a listener

2. If CFSHNM = -1 and CFSHST<>0 , this is a connect request waiting for a reply.

All other combinations indicate an unused entry.

The connection manager also maintains a table of previously seen hosts, OLDTAB. Each time a connection to another processor is lost, the processor's serial number is placed in OLDTAB. Then, each time a new connection is established, the processors search OLDTAB to determine whether each has ever been connected to the other. If both processors believe that each has seen the other, then the reconnection cannot be honored because the CFS resources on the processors have not been correctly coordinated. One of the processors, therefore, will have to crash.

In order to allow for expected KLIPA reload conditions, the monitors will "pause" whenever there is a configuration change. During the pause time, no CFS resources may be acquired or change state. Therefore, if a reconnection happens during the pause time, neither system will have to crash. The pause time is fifteen (15) seconds.

2.4 Initialization

CFS manages its own buffer pool. It creates the buffer pool in its own section, CFSSEC.

CFS creates the buffer pool by touching pages and then locking them into memory. Therefore, CFS initialization must proceed in process context.

The other side of CFS initialization is that done when SCA initializes. CFSSRV is called at CFSINI by SCA when SCA initializes. However, if CFSCSC has not yet been called, CFSINI cannot proceed to make connections and "join" the CFS network.

CFSJYN is the routine called to complete initialization and to join the CFS network. CFSJYN will only proceed the second time it is called. That is, it will proceed only after both CFSINI and CFSCSC have been called. This insures that both SCA is initialized and the CFS buffer pool has been created.

3.0 Extensions to existing services

Many of the monitor services have been "extended" to allow

for distributed control. In all cases, the code changes have been inserting calls to routines in CFSSRV and, possibly, some reorganization to eliminate a dependency on NOSKED as a system-wide interlock. For the most part, the existing monitor services no little about the workings of CFS except that it is a means for acquiring a global interlock or global resource.

The bulk of the changes have been to PAGEM and PAGUTL (PAGEM in pre-release 6 monitors). The next most significant set of changes were to DSKALC to allow for managing the bit table. Finally, there are scattered changes in various monitor routines, including MSTR, DISC and MEXEC and DIRECT.

3.1 File system changes

Each OFN has acquired some new state. The word, SPTO2 contains CFS-specific flags as well as a file access state field. The file access state field reflects the value of the file's state in the CFS data base and is stored in SPTO2 as well for convenience and efficiency.

Each opened file has at least one, and perhaps two, CFS resources assigned on the accessing processor. In addition, each active OFN has a CFS "access token" assigned as a CFS resource. Therefore, an opened file has one or two CFS file resources and at least one access token associated with it.

The file resources are:

1. file open token. This is defined by a qualifier word of the XB address. This specified whether the file is opened frozen or thawed.

2. frozen writer token. This is defined by a qualifier word containing the XB address + <DSKAB_1>. This resource is present whenever a file is opened for frozen write access on this processor. This is an exclusive resource and therefore may be owned by at most one processor.

The OFN resource is:

The file access token. Each active OFN has one such resource assigned. It is defined by a qualifier word containing the file XB + <DSKAB_3>. The access of the file access token defines the types of references that processes on the owning processor may make. Therefore, if the access token specifies "read only", then only read references may be made to the data in the file section.

The access specified in the access token is also present in SPTO2 is the field SPTST. This is so because many of the file and memory management routines, e.g. the page fault

handler, need to insure that the access is proper for the operation in question. Locating the CFS resource each time would be too costly, so the CFS state is also kept in a per OFN data table. In order to minimize the chance of error, only the routines in CFSSRV set and change SPTST.

The access of an OFN is set or changed by the routine CFSAWT or CFSAWP. The former sets the proper access but does not "reserve" the resource to this processor; the latter both sets the access and reserves it. CFSAWT is the most commonly used of the entry points. CFSAWP is used by code that needs to make many references over a period of time and does not wish to incur another page fault should the access be removed during the interval. Examples of this are:

1. During file opening

2. bit table lookup and modification

3.2 Directory locking

Directory locks are now CFS resources. The old LOKTAB and associated storage are gone. Each time a directory is locked, a CFS resource is created or modified.

The routine CFSLDR is called to lock a directory, and CFSRDR is called to unlock a directory. Directory lock resources are an example of long resource blocks.

A directory lock resource is defined as:

1. root code = structure name

2. qualifier = DRBASE + <directory number>

3.3 Directory allocation information

The remaining allocation of a directory is cached in memory to aid in processing page faults and file page creation.

Since the data is not part of a file, it has to become a CFS resource.

For each active directory, e.g. there is a directory allocation entry, there is a CFS resource defined by:

1. root = structure name

2. qualifier = DRBAS0 + <directroy number>

The code that looks up and checks directory allocations now calls CFSSRV to insure that the resource is held on this processor. The routine CFSDAU is used for the various operations on the directory allocation resource.

The optional data fields in this resource carry the value of the remaining allocation.

3.4 Bit table management

The global interlock for a bit table is the file access resource itself. When a system wishes to gain exclusive access of the bit table, it simply calls CFSAWP requesting write access to the bit table file. Write access is translated into "exclusive" access in the CFS resource.

Once the access token is reserved with write access, no other processor may gain access to the resource.

This interlock was accomplished this way for two reasons:

1. In order to modify the bit table, the processor must have exclusive access anyway, so at some point the bit table file's access token would have to be set to exclusive access

2. This avoids unnecessarily long delays in processing page faults on the bit table, and might even avoid a page fault altogether.

3.5 file token management

The file access token is the most interesting of the CFS resources, and the most important as well. It is interesting because of the complexity required to manage it and of the amount of asynchronous activity dedicated to this management.

Appendix A is a long, informal description of file resources with particular emphasis on the file access token. Rather than duplicate appendix A here, this section will cull the salient points in an effort to describe the actions required to manage the token.

If a system is required to give up access to a file section, or to downgrade its present access, DDMP must run. This operation occurs during the vote processor's operation and is triggered by the "vote OK" call-back. Note that this call-back also changes the reply from "unconditional yes" to "delay yes".

For a downgrade operation, i.e. from write to read, DDMP must run to write any modified pages to the disk. Once this is completed, CFSFOD is called to send the "cancel delay" message.

If the processor must give up all access, i.e. the other processor requires write access, DDMP must run to write all modified pages to disk and to delete any copies of the file data in local storage. Once this is completed, CFSFOD is

called as above.

While DDMP is operating on behalf of CFS, the bit SPTFO is set in SPTO2 to indicate that no accesses should be honored for any other fork. This bit is honored by the page fault handler.

3.6 Structures

Structure mounting is included in the operations managed by CFS. This is necessary in order to coordinate access to the structure by the various CFS processors.

CFS requires that each mounted structure be mounted with the same access by all accessing processors, and that the structure have the same "alias" name on all of the accessing structures. These are accomplished by creating the following CFS resources:

1. Structure name. This is a resource defined as follows

    a. root = structure alias

    b. qualifier = STRCTN

    C. unique code = drive serial number

2. drive serial number. This is a resource defined as follows:

    a. root = drive serial number

    b. qualifier = STRCTK

    c. unique code = alias name

These two resources, if successfully acquired, meet the guarantees stated above. The access type, that is whether the structure is shared or exclusive, is controlled by the DSN resource only. The name resource is always created with full sharing.

When a structure's access type is changed, that is from shared to exclusive or from esclusive to shared, the routine CFSSUG need only change the DSN resource.

Since CFS matches a structure with a DSN, it is important that if the structure is moved to another drive that the CFS resources be renamed. This is accomplished by having PHYSIO call CFSSRV at CFSRDN describing the old and new UDB for the disk pack. CFS will then update its data base to reflect the new conditions.

4.0 Operational aspects

The implementation of a multi-processor system implies some operational controls outside of the file system. In particular, the system date and time must be coordinated and kept synchronized among the processors.

This is accomplished by means of a CFS message sent to any newly added node. The opcode for the message is .CFTAD. As a result of this, a system may discover the date and time even though its front-end processor does not know the current date and time.

Should CFSSRV receive a date and time message, it simply stores it in a predefined location and waits for "job 0" to discover it. The current system date and time is changed by job 0 only.

5.0 Temporary features

Because CFS is only one part of the LCS product, and because other necessary pieces are not yet available, several "features" have been provided to smooth the transition from a single processor file system to a distributed file system. As the other LCS services become available, these temporary services will be superceded.

5.1 ENQ/DEQ

In order to avoid a malfunction of a program performing simultaneous update coordination with ENQ/DEQ, there is a temporary CFS resource representing the ENQ on a file.

Each time an ENQ file resource is first requested (i.e. an ENQ lock block is created), CFSENQ is called to register an exclusive CFS resource for the file. The resource is:

1. Root code = structure name

2. qualifier = XB + <DSKAB_2>

If the requesting processor succeeds in creating the resource, then the ENQ will be allowed. If the processor cannot create the CFS resource, i.e. some other processor indicates "no", the ENQ is denied.

.page
Appendix A

CFSSRV is a lock manager. The locks it manages represent resources in the system, but CFSSRV is not aware of the mapping of lock to resource. The mapping, or meaning, is made by the creator of the resource.

Files are a resource with CFS locks. Each file has the following CFS locks:

. open type

. write access

. ENQ/DEQ lock

In addition, each of the sections of the file, represented by an OFN, has an access token. Therefore a file has up to 512 access tokens.

When a file is opened, the "open type" and "write access" lock are acquired. The "open type" is either"

. shared read (frozen)

. shared read/write (thawed)

. exclusive (restricted)

. promiscuous (unrestricted)

The word in parentheses represents the argument to OPENF%.

If the opener requests "frozen write" access, then if the "open type" lock is successfully locked, i.e. no one has the file open in a conflicting mode, the "write access" lock is acquired. This is an exclusive lock that represents the single "frozen write" user of the file. The lock is held by the system that has the file opened "frozen write".

Each of the locks described above apply to a file, that is something described by an FDB. In addition to these, each file has some number of OFNs, one for each file section that is in use. Therefore, a file may have up to 512 OFNs or file sections.

Each active OFN has an "access token" lock. The access token represents the ability of the system to access the data described by the OFN. The access token may be held in one of the following modes:

. place-holder

. read-only

. exclusive (read or write)

A read-only access token may be held by any number of systems simultaneously. An exclusive token is held by only one system. A "place-holder" access token is an artifact that permits the CFS systems to agree on the end-of-file correctly. It also has some ramifications for bit table access tokens that will be described later. Place-holder tokens are also an optimization to avoid reallocating tokens that have been "lost" to another system.

The file access token is the most fundamental CFS lock in

that it is used not only to control simultaneous access to user files, but also to manage directories and bit tables.

The access token state transition table is given below, with the action required to make the designated state change

old \ new	read	exclusive	place-holder
read	nothing	vote	DDMP*
exclusive	DDMP**	nothing	DDMP*
place-holder	vote	vote	nothing

Where:

vote means that the other CFS systems must be asked for permission to make the state transition. Voting is a fundamental operation of CFSSRV and is done by a software implemented broadcast.

DDMP* means that DDMP must run and remove all of the OFN's pages from memory and update the disk copy of any modified pages.

DDMP** means that DDMP must run to update to disk any modified pages and any in memory pages must be set to "read only". This latter operation is performed by clearing the CST write bit. The CST write bit has been implemented in KL paging explicitly to support loosely-coupled multi-processors.

While DDMP is performing a CFS-directed operation, all pages of the OFN are inaccessible to any other process. This is achieved by a bit, SPTFO, set in SPTO2 by DDMP.

Access permission to a file moves among the CFS systems on demand. Each system must remember its state of the token so it may respond to requests for the access permission.

The token consists of:

- The structure name
- the OFN disk address
- a flag bit to indicate this is the access token
- state

. end-of-file pointer

. end-of-file transaction number

. fairness timer

. the OFN this token is for

and, if this is a token for a bit table:

. structure free count

. structure free count transaction number

The fairness timer is a CFS service that allows a resource to be held on a node for a guaranteed interval. Therefore, the owner need not lock the resource and arrange to unlock it later. Rather it simply places the guarantee interval in the resource block and the CFS protocol takes care of the rest.

Place-holder tokens exist principally to hold the values associated with the end-of-file pointer and with the structure free count. It is important that these be held by each system, because the owner of the OFN token may crash and therefore the last known state of these quantities must be remembered so that the remaining nodes may have the best possible value for them. The transaction count is intended to determine whose value is the most recent should the owner not be present to contribute the current value. During the voting for acquiring a token, these values are passed among the CFS nodes, and the node conducting the vote retains the values associated with the largest transaction number.

The file access token represents the rights that a system has to access a file section. That is, the token is associated with the file's contents.

However, the owner of a file, i.e. the system holding exclusive rights to access the file, also has the right to modify the file's index block. The owning system may add pages to the file or delete pages from the file.

OFNs are treated specially in TOPS-20. Unlike the file's data pages, an OFN may not be discarded when the system gives up its access to the file and read from its home on the disk when the access is reacquired. An active index block, represented by an OFN, contains paging information that must be retained while the file is opened. For this reason, a system needs to be informed if the index block contents are changed by another system.

This information is disseminated in CFS by a broadcast message. Each time a system writes a changed index block to disk, it informs all of the other CFS systems by a broadcast

message. Note that this broadcasting is done only when the changed index block is written to disk, and not each time the index block is modified. A broadcast message is used instead of including this in optional data with the access token for reasons explained in a later section of this document.

When a CFS system receives such a message, it sets a status bit in the apporpriate OFN so that the next time a process attempts to reference the OFN the following will happen:

. the disk copy of the index block is examined.

. for each changed entry, update the local OFN

This reconciliation of the index block with the local OFN is accomplished by the routine DDXBI.

RESOURCE ACQUISITION AND UPDATING

CFS resources are acquired and changed in response to requests from other parts of the monitor. Rather than describe each one, it will be instructive to consider how the file related resources are acquired, maintained, and destroyed.

When a file is opened, and the first OFN is created, ASOFN will create the static CFS resources: open type and, if appropriate, the frozen writer token.

Anytime an OFN is created, be it in response to opening the file, or one of the "long file" OFNs, ASOFN will create the access token.

The access token state is verified by various of the file system and memory management routines. The most common place for this is in the page fault handler. The two exceptions to this are for a bit table access token and a long file "super index block". The bit table token is acquired and "locked" when the bit table lock is locked and released only when the bit table is unlocked. The token for a super index block is occasionally acquired in DISC by the routine (NEWLFT) that creates new long file index blocks. In theory, these exception cases need not be exceptions. That is, the code could simply rely on the normal management of the token during page faults to insure data integrity. However, in these cases, the code must perform multiple operations on the file data "atomically". That is, it must modify two or more pages, or it must "test and set" a location with the assurance that no other accesses to the data occur between the steps. On a single system, this is done by a NOSKED to prevent any other process from running. In an LCS environment, NOSKED is not sufficient (although it is necessary!). Another form of interlock must be used to prevent a process on another system from examining or

modifying the data. It turns out that the access token satisfies this need quite well.

The above discussion implies that the page fault handler, when it acquires an access token for an OFN, does not "lock" the token on the system. That is, the token is acquired but not "held". This may result in the token being preempted by another system before the process is able to reexecute the instruction that caused the page fault. The "fairness" timer in the token resource is one attempt to minimize such thrashing.

The access token is acquired on the following conditions:

- when an OFN is being created
- when the OFN is locked
- when a page fault occurs because the current access is not correct

The current state of the token is kept in the CFS resource block as well as in the OFN data base. The field, SPTST, is the current OFN state of an OFN. The values are:

0 => no access

.SPSRD => read only

.SPSWR => read/write

SPTST is modified by the routines in CFSSRV that are called to set the state of the file. The values are set here, and not in PAGEM, PAGFIL or PAGUTL because the OFN state must be set while the CFS resource block is interlocked against change.

The routines to modify the state of an OFN token are:

- CFSAWT - acquire token but don't hold it
- CFSAWP - acquire token and hold it

TOKEN MANAGEMENT

Once a token is "owned" on a system, it will remain in that state until it is required on another system. That is, if the token is held for read/write access (exclusive), then all references to the pages of the OFN will succeed without CFSSRV being invoked.

If a token must be revoked because another system needs it, CFSSRV signals DDMP to process the data pages. This is done by:

. Setting bits in the field STPSR in the OFN data base.

. Setting the OFN's bit in the bit mask OFNCFS.

. Waking up DDMP.

The field STPSR is a two-bit quantity indicating the type of access required by the requesting system. DDMP's action is as follows:

read-only needed:

Write all modified pages to the disk. Clear all of the CST write bits in all in-memory pages.

read/write needed:

Write all modified pages to disk. Flush all "local" copies of data including any copies on the swapping space. Swap out the OFN page if it is in memory (actually, simply place it on RPLQ).

Once DDMP has performed the necessary operation, it calls CFSFOD. This routine will set the OFN state and the resource state appropriately as follows:

read-only requested:

set OFN state to .SPSRD and set resource state to "read".

read/write requested:

set OFN state to 0 and set resource state to "place-holder".

CFSFOD also copies the current end-of-file information from OFNLEN into the resource block and finally it sends the "condition satisfied" message to the requestor.

While DDMP is performing its work on behalf of CFS, it sets the bit SPTFO in the OFN data base. This bit is examined by the page fault handler, and by CFSAWP/CFSAWT to see if the OFN is in a transition state. If SPTFO is set, and the process requiring the OFN is not DDMP, then the process is blocked until SPTFO is cleared by DDMP. In order to facilitate identifying DDMP from all other processes, a new word has been added to the PSB called DDPFRK. If DDPFRK is non-zero, then the current process is indeed DDMP and SPTFO should be ignored.

JNUSED RESOURCES

Whenever a node replies "no" to a request, it remembers in

the associated resource block the node(s) that have been rejected. The only reason for unconditionally denying a request is that the resource is "held" locally. If a resource cannot be granted because of the fairness timer, the "no" response includes an optional data word of the time the resource is to be held. Therefore, the requestor knows precisely when to request the resource anew.

When a held resource is "released" (or undeclared), CFS examines the rejection mask for the resource. For each node identified in the mask, a "resource released" message is sent indicating that this is a propitious time to try to acquire the resource. There is no guarantee the new request will be granted as the resource could be held again, or another node could have requested, and been granted, the resource first.

DELETING FILE RESOURCES

The access token is deleted whenever the associated OFN is deassigned.

The static file resources are released when the file is closed. This is performed in RELOFN.

CHANGES TO EXISTING CONCURRENCY CONTROL SCHEMES

As a result of CFS, much of the concurrency control in TOPS-20 has become distributed. In some cases, this has been done by creating a companion resource to an already existing one. As example of this is the file open mode resource described above.

In other cases, existing locks have been replaced by CFS resources.

The decision as to which technique to employ was made on a case-by-case basis. The significant criterion was how easy it was to eliminate the existing concurrency control and replace it with the CFS management. The file resources proved difficult to do. However, there are two important pieces of the monitor's structure that were easily and efficiently replaced: directory locks and directory allocation tables.

Directory locks are now CFS resources. A directory lock resource contains:

- the seventy-two bit identifier
- owning fork
- access type

. share count

. waiting fork bit table

In fact, a directory lock resource is the sole instance of a "CFS long block".

Directory locks are always acquired for exclusive use. However, unlike file access tokens, directory locks are never granted "conditionally". This is because directories are files, and the directory contents are subject to negotiation by the associated file access token. That is, acquiring exclusive use of the directory lock resource is independent of acquiring permission to read or write the directory contents. When some process on the owning system attempts to read or write the directory contents, it must first acquire the file access token in the proper state. Although this sounds somewhat inefficient, i.e. requiring the node to acquire two independent resources, it is in fact a remarkably efficient adaptation of the CFS resource scheme. This is so because a node need not know how the directory contents will be used when it acquires the directory lock. That is the way the lock was handled before CFS, and preserving this convention means that the code to acquire the directory lock under CFS is as efficient as possible. The state of the file access token, and consequently the degree of sharing of the directory contents, is determined by how the contents are referenced and not by how the directory is locked. This means that a process may lock the directory lock without knowing how it will reference the associated data, and its reference patterns determine what other negotiations are required.

The directory allocation table is a local "cache" for the information normally stored in the directory. Each active OFN is associated with a directory allocation entry. Each entry is for exactly one directory. The entry, before CFS, contained: structure number, directory number, share count, and remaining allocation.

Under CFS, an active allocation entry contains: structure number, directory number, share count, and pointer to the CFS resource block. The CFS resource block contains, besides the normal CFS control information, the remaining allocation for the directory and a transaction number. The transaction number serves the same purpose as the transaction number associated with a file end-of-file pointer.

CFS may have an "unused" resource block for a directory allocation entry. That is, even though there is no active directory allocation entry, there may be a CFS resource block representing the directory. This is because CFS attempts to retain knowledge of resources for as long as possible to avoid having to vote when some process wishes to

create the resource anew. However, CFS will destroy any unused resource allocation entry that is requested by another system.

TRANSACTION NUMBER

The optional data items, "end-of-file pointer" and "structure free space", have an associated value called the "transaction number".

One either uses centralized or decentralized control in a "loosely-coupled multiprocessor" system. In a centralized system, control information and updating is coordinated by a master. Transactions are "serialized" by virtue of having a single owner for the resoruce and therefore a single manager of the resource data. In a decentralized system, the various systems share the ownership of resources and use some sort of "concurrency control" technique to manage resources.

CFS is a decentralized system. A resource is not owned or managed by any particular system, but rather the responsibility for the resource is passed from system to system as required. As such, it may not always be possible to uniquely identify a particular system as the owner. This may cause a problem when a system needs to become the owner, and therefore must determine the current status of the resource in question.

There are two possibilites that a nascent owner may encounter:

. The previous owner is present and indentifiable.

. There is no system that is the previous owner

and of this latter case:

. the existing control information is accurate

. the existing control information in not accurate.

Clearly, if the previous owner is present, the new owner has all of the information it needs to proceed with its transaction.

If the previous owner cannot be identified, then the new owner must be able to determine which of the systems has the current control information about the resource. It may be that none of them has, and this is a problem that exists even on a single-processor system. The result of such a problem may be "lost pages", inconsistent data bases and other such phenomena. As in a single-processor system, the problem occurs because the resource control information is lost as an effect of a system crashing.

In order to determine the most up-to-date information about a resource, each system maintains a transaction count along with the information. Whenever it acquires information with a larger transaction count than its own value, it knows that information is more current and it must replace its own copy with the new data and count. Whenever a system unilaterally changes its copy of the control information, it must also increment the associated transaction count. Since a system may perform such an update only when it has write or exclusive access to the resource, the system need change the transaction count only when it must downgrade its access.

Due to the nature of the CFS voting and resource management, it is possible for a system to acquire a resource but to receive a different value for the resource control information from each of the other systems (this will happen only if the owner crashed. If the owner didn't crash, then at least two of the other systems must have the same control information and transaction count). In this case, the transaction counts are used to identify the most up-to-date value.

The transaction count is really a "clock" that is used to "time-stamp" information. When systems communciate with one-another, they synchronize the clocks by sending each other the current counts. Most network concurrency schemes use clocks for similar purposes, and most of the uses and implementations are considerably more exotic than this one. However, since CFS needs the clock only to determine relative ages, and not absolute ages, of information, this simplified clock is adequate.

An alternative to using transaction counts is to "broadcast" changes to resources. This has the disadvantage that it is costly in both processor and communications time and resources. However, CFS does use broadcasting in a few cases where the lack of up-to-date information could result in data being destroyed. The two cases are:

- an OFN being modified and written to disk
- an EOF value being written into the directory copy on the disk

As both of these represent changes in the permanent copy of the resource, it is essential that all of the other systems have current copies or knowledge of the update.

CFS MESSAGE SUMMARY

Items marked with a "*" are sent as broadcast messages.

1. request resource (vote)

2. reply to request:

a. unconditional yes

b. unconditional no

c. no with retry time

d. conditional yes

3. resource available

4. condition satisfied

*5. OFN updated

*6. EOF changed

In addition, each message type may carry specific optional data items, up to four words of optional data per message.