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.

VOTING IN CFS

When a node needs to "upgrade" its access to a resource, including acquiring a new resource, it must poll each of the other CFS nodes. This is so because none of the CFS nodes is a master and therefore there is no a priori location for resolving access requests. CFS is not only a democracy, but somewhat of a cacophony.

Voting, then, requires "broadcasting" to each other node the required resource and access. Each node must respond with its permission or denial.

The CI does not support broadcast, and even if it did, it would not support a reliable broadcast. Therefore, CFS implements broadcasting by sending a message to each of the other nodes, one-at-a-time.

A vote request contains:

- function code
- resource "name" (seventy-two bits)
- access desired
- vote number

A reply contains:

- function code
- resource name (seventy-two bits)
- reply (yes, no or "qualified yes")
- vote number
- optional data

The message contains a function code because votes and replies are only one kind of CFS to CFS communication.

The vote number is used to insure that the reply is to the proper request. The requestor may "restart" a vote at any time. It does this be "canceling" the current vote, acquiring a new vote number, and broadcasting the new request. A vote number is a monotonically increasing, thirty-six bit quantity.

A vote will be restarted for one of the following reasons:

- a configuration change is reported by SCA
- a previous vote "times out".

The latter should rarely occur, and is likely indicative of a malfunctioning CFS on some other system. In some cases, a node will not reply if it is unable to acquire the appropriate space for constructing a message. There are a small number of cases where this is legal, and for these cases, the requestor must revote when appropriate.

When a reply is received, the vote number must match the number in the associated resource block.

The replies to a vote are:

. unconditional yes.

. no

. conditional yes.

. cancel yes condition

A conditional yes means that the respondent will approve the request, but it needs to perform a local housekeeping operation first. The most common form of this is voting for an access token where the respondent must first update the disk copy of the file, and perhaps flush all of its local copies of the file data. When the condition has been satisfied, a "condition satisfied" reply is sent.

Each resource has a "delay mask". This mask has a bit for each of the other CFS nodes, and whenever a node replies with "conditional yes", its bit is set in the resource's delay mask. Therefore, a process that is waiting for the conditions to be satisfied, simply examines the delay mask periodically and waits for all of the delay bits to be cleared. While any delay bits are set, the vote is considered to be still in progress, and therefore any configuration change will require restarting the vote.

Conditional yes votes, and the associated delay mask, are provided to eliminate the need for nodes to reply "no" when there are temporary conditions preventing the approval of the request. The overhead required to process such replies, and to wait for them, is offset by the gains in not having to revote in the face of such conditions.

CFS provides the following basic voting services:

. Acquire a resource. If the resource is known on this node, but the current state conflicts with the request, the currently held resource is released and a vote is taken.

This service is called specifying either "retry until successful", or return after one try.

. Upgrade a resource. This service tries only once. It also guarantees that the currently held resource will not be released. In fact, the resource may be held and "locked" locally when "upgrade" is requested.

. Acquire local resource. This is used for resources not shared by other CFS nodes, but managed by CFS. Examples are directory locks on exclusive structures.

VOTE MECHANISM

A vote is started by the routine VOTEW. Ordinarily, one does not call this routine directly, but rather one requests a resource, and if necessary, VOTEW will be called to conduct a vote.

VOTEW always waits for the vote results. The results are tallied at interrupt level by noting the number of replies received in the associated resource block. VOTEW periodically examines the resource block testing for:

- all tallies received
- a "no" vote recorded
- a configuration change

The actions taken are as follows:

- configuration change: restart the vote
- a "no" vote: return to the called
- all tallies received:
  - if no "conditional yes" votes, return to caller
  - If one or more "conditional yes" votes, wait for the "condition satisified" replies. While waiting, a configuration change could occur, requiring the vote to be restarted.

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.

UNUSED 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.