CFS Resources

David Lomartire

7-Nov-84

================

1. Files - An opened file has one or two CFS file resources (file open token and possibly frozen writer token) and a file access token for each active OFN (at least one). The field SPTST of SPTO2 holds the access specified in the file access token. While DDMP is performing CFS-directed operations on a file, all pages of the OFN are inaccessible to any other process. This is achieved by a bit SPTFO, set in SPTO2 by DDMP. FKSTA2 will contain the resource block address (when appropriate) of the CFS resource the fork is waiting on.

Below is the table of contents fragment from CFSSRV which illustrates the various file related routines:

File Open Token

CFS routines:	CFSGFA - Acquire file opening locks
	CFSFFL - Release file locks
	CFSURA - Downgrade to promiscuous (unrestricted, OF%RDU)

When the file is opened via OPENF%, it will have one of the following open types assigned to it:

Open type (spec)	CFS term (code)	OPENF% term	OPENF% bits
shared read	.HTOSH - read-only shared	Frozen	not OF%THW
shared read/write	.HTOAD - full sharing	Thawed	OF%THW
exclusive	.HTOEX - exclusive	Restricted	OF%RTD
promiscuous	.HTOPM - promiscuous read	Unrestricted	OF%RDU
local exclusive	1B0!.HTOEX - local exclusive	--	OF%DUD

** CFSGFA - Acquire file opening locks **

Called by: GETCFS in PAGUTL

Upon entry to CFSGFA, the access type is converted into one of the access codes shown above. Next, HSHLOK is called to see if a file open token already exists for this file. If it does, a call is made to CFSUGD to upgrade the already existing access to the new access which is requested.

If a file open token does not already exist, CFSSPC is called to get a short request block. CFSSPC will return a block with HSFLAG, HSHPST, and HSHOKV zeroed as well as HSHRET set to 1,,SHTADD. Then, the following is placed in the block:

HSHROT	six-bit structure name
HSHQAL	index block address
HSFLAG	HSHTYP=access, HSHLCL set if local exclusive

Finally, CFSGTT is called to get the token (with "try only once" set). (Note, if the structure is set local exclusive, CFSGTT will discover this and use CFSGTL. This will mean that HSHVTP will not be updated with the access of the vote since no vote is required.) If the token is acquired, the following has been updated in the resource block:

HSFLAG	HSHVTP=access of vote, HSHCNT incremented (owned)
HSHFRK	FORKX of running fork
HSHTIM	TODCLK stamp when vote approved and token obtained

If, upon return from CFSGTT, we do not need the newly created block (T1 is non-zero), SHTADD will be called to return the extra block to the CFS pool.

** CFSFFL - Release file locks **

Called by: FRECFS in PAGUTL

The routine CFSFFL simply calls CFSNDO to release the file open token. See the discussion of CFSAWT/CFSAWP for a description of CFSNDO.

** CFSURA - Downgrade to promiscuous (unrestricted, OF%RDU) **

Called by: RELOFN in PAGUTL when OFOPC in SPTO2 is 0
(no more "normal" (non-unrestricted) openings)

The routine CFSURA is called by PAGEM to downgrade the access of a file open token to .HTOPM whenever all open OFNs are closed. It calls CFSUGD with a new access of .HTOPM and decrements HSHCNT when the new access is obtained.

Frozen Writer Token

CFS routines:	CFSGWL - Get write access
	CFSFWL - Free write access

If a file is opened for frozen write (OF%WR and not OF%THW), then the frozen writer token is acquired after the file open token is obtained. This is an exclusive access token that represents the single "frozen write" user of the file. It is held only by the system which has the file open for frozen write.

** CFSGWL - Get write access **

Called by: CHKACC and GETCFS in PAGUTL

Upon entry to CFSGWL, a short resource block is obtained via a call to CFSSPC. CFSSPC will return a block with HSFLAG, HSHPST, and HSHOKV zeroed as well as HSHRET set to 1,,SHTADD. The following is then placed in the block:

HSHROT	six-bit structure name
HSHQAL	FILEWL+index block address
HSFLAG	HSHTYP=.HTOEX

Finally, CFSGTT is called to get the token (with "try only once" set). (Note, if the structure is set local exclusive, CFSGTT will discover this and use CFSGTL. This will mean that HSHVTP will not be updated with the access of the vote since no vote is required.) If the token is acquired, the following has been updated in the resource block:

HSFLAG	HSHVTP=access of vote, HSHCNT incremented (owned)
HSHFRK	FORKX of running fork
HSHTIM	TODCLK stamp when vote approved and token obtained

If, upon return from CFSGTT, we do not need the newly created block (T1 is non-zero), SHTADD will be called to return the extra block to the CFS pool.

** CFSFWL - Free write access **

Called by: RELOFN and FRECFS in PAGUTL

The routine CFSFWL simply calls CFSNDO to release the write access token. See the discussion of CFSAWT/CFSAWP for a description of CFSNDO.

File Access Token (OFN access token)

CFS routines: CFSAWT/CFSAWP	- Acquire/Acquire and reserve access token	
CFSUWT	- Release access token	
CFSFWT	- Free write token	
CFSDWT	- Write token revoked	(callback)
CFSOVT	- Approve sharing of OFN	(callback)
CFSDAR	- Optional data for access token	(callback)
CFSBOW	- Broadcast OFN change	
CFSBEF	- Broadcast EOF	
CFSFOD	- DDMP force out done	

Each active OFN has an access token. It may be in one of the following modes:

* place-holder - .HTPLH (this value must be zero!)
* full sharing - .HTOAD
* exclusive (read or write) - .HTOEX

The location CFSOFN points to a table which is NOFN long and is indexed by OFN. It contains the address of the resource block which describes that OFN.

** CFSAWT/CFSAWP - Acquire/Acquire and reserve access token **

Called by:

NEWLFP	in DISC	for	.HTOEX access	(CFSAWP)
UPDLEN	in DISC	for	.HTOEX access	
GETLEN	in DISC	for	.HTOAD access	
MAPBTB	in DSKALC	for	.HTOEX access	(CFSAWP and CFSAWT)
RELMPG	in PAGEM	for	.HTOAD access	(CFSAWP)
NTWRTK	in PAGEM	for	.HTOEX access	
NIC	in PAGEM	for	.HTOEX access	
OFNTKN	in PAGUTL	for	.HTOAD access	
DDXBI	in PAGUTL	for	.HTOAD access	(CFSAWP)
UPDPGS	in PAGUTL	for	.HTOAD access	(CFSAWP)
ASGOFN	in PAGUTL	for	.HTOAD access	(CFSAWP)
LCKOFN	in PAGUTL	for	.HTOAD access	(CFSAWP)
MRKOFN	in PAGUTL	for	.HTOAD access	(CFSAWP)

The routines CFSAWT and CFSAWP acquire the access token. The latter leaves the resource block reserved on the system. The former does not.

Upon entry, the access type is checked. If zero, full shared (.HTOAD) access is acquired. If not equal to zero, exclusive (.HTOEX) access is acquired. Next, the SPTFO bit in SPTO2 is checked to see if DDMP is forcing this OFN to disk. If so, the fork goes into WTFOD wait. Otherwise we proceed to lookup in the CFS OFN table (pointed to by CFSOFN) the address of the resource block for this OFN. If none exists (entry is zero), we continue at CFSAW1 and add an entry.

At CFSAW1, GNAME is called to get the structure name. Then CFSSPC is called to obtain a short resource block. CFSSPC will return a block with HSFLAG, HSHPST, and HSHOKV zeroed as well as HSHRET set to 1,,SHTADD. The following is then placed in the block:

HSHROT	six-bit structure name
HSHQAL	FILEWT+index block address
HSFLAG	HSHTYP=access, HSHKPH set
HSHCOD	OFN
HSHPST	1,,CFSDWT
HSHOKV	1,,CFSOVT
HSHCDA	1,,CFSDAR
HSHOP1	0 (transaction number)

Next, a call is made to get the resource. CFSGET is called if the structure is shared. If the structure is set exclusive, CFSGTL is used. (Note that if CFSGTL is called, HSHVTP will not be updated with the access of the vote since no vote is required.) The call is made to "retry until successful" so, upon the return, we have acquired the resource. The following has been updated in the resource block:

HSFLAG	HSHVTP=access of vote, HSHCNT incremented (owned)
HSHFRK	FORKX of running fork
HSHTIM	TODCLK stamp when vote approved and token obtained

Now, HSHFCT is set to be TODCLK+WRTTIM. A check is made to see if optional data was returned during the vote. The optional data will be the most recent value of OFNLEN in HSHOPT and the transaction number in HSHOP1. This value represents what the other node believes OFNLEN is for that OFN. If there is optional data present from the vote (HSHODA is set), this must be the most current value of the file length and it is stored in OFNLEN. In this case, the other node had more recent information than us so we must update our copy of OFNLEN. Otherwise, no other copy exists and we initialize the optional data of the resource block to contain the current OFNLEN entry for that OFN. HSHOP1 is incremented to initialize the transaction number. In this case, no other node had more recent information than us so we establish ourselves as the node which knows the state of OFNLEN. Note that if this access token was for a bit table, the structure free count must also be maintained or established. The callback routine CFSDAR will insure that STOFRC is called to update the structure free count if appropriate.

If CFSAWT was originally called, CFSNDO will be called to undeclare the resource. CFSNDO will decrement HSHCNT and HSHNBT will be "searched" looking for previously rejected hosts to notify (via CFNOHN). If CFSAWP was originally called, the resource will remain owned by the fork. In either case, the resource block remains in the hash table and HSHKPH will remain set. The only distinction is whether we own the resource or not.

Finally, the SPTST field in SPTO2 is set to the correct state; .SPSWR (2) for .HTOEX access tokens or .SPSRD (1) for .HTOAD access tokens.

The description above describes what occurs when CFSAW1 is entered to add a new OFN entry. However, there is another option. If CFSOFN already contains the address of a resource block, then this OFN has already been added and is known to this CFS system.

First, the post callback address (HSHPST) is checked and set to 1,,CFSDWT if it was zero. The block is then locked against removal by either CFSRSE or CFSUWT by incrementing HSHLKF. Both of these routines can remove resource blocks from the hash table. So, we are locking the resource block against possible removal from the hash table via the use of HSHLKF. Now, a check is made to see if there is anyone waiting for this block. This is done by checking HSHTWF, HSHUGD, and HSHWVT. HSHTWF gets set when the fork is going to go into a wait state on that block (like CFSRWT). The setting of STKVAR WTFLAG indicates to CFSUGW (the wait routine) if we set HSHTWF. If WTFLAG is -1, HSHTWF was already set and if it is zero, we set HSHTWF. HSHTWF is also set when we call CFSUGD to upgrade access to the token. HSHUGD is set when we are performing an upgrade vote on the resource. HSHWVT is set when we are voting on a resource.

So, if there is someone waiting on this block, we continue at CFSUGW. We pass into CFSUGW the address of the wait routine; in this case CFGVOT. In CFSUGW, we place TODCLK+^D500 in HSHTIM and call CFSWUP (the general wait routine) to wait until the vote has completed. Upon return from CFSWUP, the block will no longer be in a "wait state". We will clear HSHTWF if we had set it (WTFLAG=0) and check HSHPST to see if the block has been released (it will be zero if so). If it has not been released, we unlock it by decrementing HSHLKF, clear some bits which could be left over from voting (HSHRTY and HSHVRS) and start over again at CFSAWL to try to acquire the access token. If the block has been released, we get the value of HSHLKF and decrement it. If the resulting value is non-zero, we are not the last locker, so we couldn't obtain the access token and return without changing the access token we currently have (current access reflected in SPTO2). If the decremented value of HSHLKF is now zero, then we are the "owner" of the resource block. The block address in CFSOFN will be cleared and CFSRMV will be called to remove the block from the hash table. Again, we will return without changing the access of the OFN. CFSRMV will not post the removal.

If there is no one waiting on the resource, we can proceed to try to upgrade our access. First, we check STRTAB to see if the structure is mounted exclusively. If so, we make HSHTYP be exclusive (.HTOEX). This is done regardless of the access we were asking for because access on an exclusive structure is always .HTOEX (it needs to be nothing other because this is the only node which can use this structure). We will never be refused access to an OFN on an exclusive structure due to setting HSHTYP to .HTOEX (as shown below).

Now, we check HSHTYP to see what kind of access we hold on the resource. If is is exclusive (.HTOEX), then we are granted access. If CFSAWP was called, HSHCNT is incremented in order to hold ownership. Finally, the OFN access is set in SPTO2.

If HSHTYP is not exclusive, we will have to upgrade our access to the OFN. First we set HSHTWF to indicate we are processing this resource block. Next we call CFSUGD to try to upgrade our access. CFSUGD will modify HSHTYP and HSHCNT to reflect the state of the resource after the upgrade attempt. HSHTYP will contain the current access and HSHCNT will contain the number of owners of the resource.

If CFSUGD successfully allows the upgrade, we will clear HSHTWF and place TODCLK+WRTTIM in HSHFCT. The block will be unlocked via a decrement of HSHLKF and, if CFSAWT was originally called, HSHCNT will be decremented so that the resource will not be held. If HSHCNT is zero, CFNOHS will be called to notify any nodes which were rejected in the interim. Next, if any optional data was returned in the upgrade vote, it is placed in OFNLEN. This would be the file length for that OFN. Finally, the OFN access in SPTO2 is updated to reflect the new access state.

If CFSUGD does not allow the upgrade, HSHWTM is checked to see if a retry wait time was given. If so, it is added to TODCLK and placed in HSHTIM. Otherwise, TODCLK+^D20 is used. Processing will now continue at CFSUGW, with the wait address specified as CFSRWT. CFSRWT will awaken under one of the following conditions:

1. The block can no longer be found in the hash table (it has been released)

2. The same block is found by HSHLOK (address match) and :
    a) HSHVRS or HSHRTY is set in the resource block
    b) HSHTIM is less than or equal to TODCLK

3. A different block is found by HSHLOK and:
    a) HSHCNT is zero
    b) HSHTYP is not .HTOEX but does match the desired access

** CFSUWT - Release access token **

Called by:	FRECFS in PAGUTL

When a file is closed, CFSUWT is called to release the access token. The resource block is found (via HSHLOK) and the OFN is retrieved from HSHCOD. The SPTO2 entry for this OFN is checked to see if anyone is waiting for this access token. This is indicated by the field SPTFR being set which indicates that CFS has requested a DDMP force out to be done for that access token. (SPTFR is set in routine CFSDWT and cleared in CFSFOD.)

If SPTFR is set then a force out has been requested for this OFN. At this point we clear the bit in OFNCFS which indicates which OFN DDMP should force out and call CFSFDF to signal that the force out is done. (CFSFDF is an alternate entry point to CFSFOD which will always signal that the force out is done regardless of the number of sharers remaining on that OFN. In effect, it "forces" the force out regardless of the current number of sharers of that OFN. CFSFOD will only signal that the force out is done when there are 2 or fewer sharers indicated by HSHCNT.) (OFNCFS is a multi-word bit mask scanned by DDMP to determine which OFNs need to be forced out.) Finally, CFSUWT is continued at the beginning to try to release the access token again.

If SPTFR is zero, then this OFN is not being forced out. We call CFNOHS to notify any hosts which we rejected for this OFN. Next we check HSHLKF to see if the resource block is locked. This will be set by CFSAWT/CFSAWP to prevent the block from being removed from the hash table. If the block is locked, we simply clear HSHTYP, HSHCNT, HSHKPH, and HSHPST. By clearing HSHKPH, the block is eligible for removal as stale by routine CFSRSE. If the block is not locked, we clear the corresponding entry in CFSOFN and remove the block from the hash table via a call to CFSRMV. CFSRMV will not post the removal.

** CFSFWT - Free write token **

Called by:	DDXBI in PAGUTL
	UPDPGS in PAGUTL
	ASGOFN in PAGUTL
	ULKOFN in PAGUTL
	UMPBTB in DSKALC

The routine CFSFWT simply calls CFSNDS to release the write access token. CFSNDS is an alternate entry point to CFSNDO which will employ a "fairness test" when notifying other nodes of the resource release. It will set HSHRFF in the block just before calling CFNOHS. CFSFWT is the only routine that calls CFSNDS. See the discussion of CFSAWT/CFSAWP for a description of CFSNDO.

** CFSDWT - Write token revoked (callback) **

Called by: CFSRTV when we want to release the resource (Note: CFSRSE has the ability to call a post routine but it should never be called for a file access token since HSHKPH will be set and this will prevent the block's removal. In fact, if CFSDWT were to be called, incorrect or needless DDMP action could result.)

This routine is the callback routine placed in HSHPST when the access token is formed. To post removals, CFSRMX is called. CFSRMX is the alternate entry point to CFSRMV used to do posting of removals. CFSRMX will insure that CFSDWT is called to do any cleanup that is needed before the resource block is removed from the hash table.

CFSDWT simply invokes DDMP to force out the pages of the OFN being released. If the resource is a place holder, then nothing is done and CFSDWT just returns. Otherwise, SPTFR is set in SPTO2 for the OFN. Note that SPTFR is a two bit field (bits 22 and 23) so both bits will be set. Next, if we own the resource exclusively (HSHTYP) and the vote request is for any access other than exclusive (HSHVTP), we will clear bit 22, which is named SPTSR. If we do not own it exclusively (we own it .HTOAD) then SPTSR will remain set.

SPTSR is checked by routine DDOCFS (in PAGUTL) when deciding what to do with the copy of the pages. If SPTSR is zero, then the vote request was not for exclusive access (and we had .HTOEX access) so UPDPGY is called to update to disk any modified pages and set any in memory pages to "read-only" (via the use of the CST write bit). (In the access token state transition table in the spec, this is shown as DDMP**.) This will result in a new access for the OFN of .HTOAD (full sharing) on the processor which used to own the resource exclusively. If SPTSR is set, then the vote request was for exclusive (or we only had .HTOAD access) so UPDPGX is called to update to disk any modified pages and remove all the OFN pages from memory. (In the access token state transition table in the spec, this is shown as DDMP*.) This will result in a new access for the OFN of .HTPLH (place-holder) on the processor which used to own the resource exclusively.

Finally, DDCFSF is incremented to wake up DDMP and the appropriate OFN bit in the OFNCFS bit-mask is set to indicate to DDMP which OFN to force out. Also, T1 is set to zero. This is important because, upon return to CFSRMX, T1 is checked and, if zero, a -1 will be placed in T1 upon return to CFSRTV and the block will not be removed from the hash table. This value in T1 is taken to be the vote type which is placed in .CFTYP. So, this is how a -1 (or conditional yes) is generated; namely, a vote comes in which causes an access token to be released and requires DDMP to run. When this is done, CFSFOD will send a "condition satisfied" message (.CFTYP = -2) to indicate that the force out is done. (The appropriate HSHDLY bit for the node is set when the -1 is received and cleared when the -2 is received. This is done in CFSRVT.)

** CFSOVT - Approve sharing of OFN (callback) **

Called by: CFSRTV when vote is to be approved

This routine is placed in HSHOKV when the access token is formed. The routine is called when the vote is to be approved in order to place this node's optional data in the vote packet.

If HSHOP1 is non-zero, then this node has some copy (it may be old) of the file length information (OFNLEN for that OFN). Both HSHOPT (the file length) and HSHOP1 (the transaction number) are placed in CFDAT and CFDT1 of the CFS send packet. Also, the CFODA flag will be set to indicate that optional data is present. If HSHOP1 is zero (no transaction number) then this node has no optional data to contribute concerning the file length and CFDT1 (the file length transaction number) is set to zero. This is done because there is also a structure free count which can be sent. So, just using CFODA is not enough. However, sending a transaction number of zero will insure that processing of the file length data will be ignored.

Finally the HSHBTF flag is checked to see if this is a bit table OFN. If it is, SNDFRC will be called to set up the send packet with the structure free count data. SNDFRC will call GETFRC and place the returned value of the free count in the CFDST0 of the send packet. Next, the current transaction number for the structure free count is retrieved from the CFSSTR table. This table, indexed by structure number, contains the transaction number values for each structure. This transaction value is placed in CFDST1 and, if we are the exclusive owner of the OFN, it is incremented. This will insure that the count is updated on the remote system. Finally, CFODA is set to indicate there's optional data present in the vote packet.

This optional data information will be processed by the CFSDAR callback routine (described below).

** CFSDAR - Optional data for access token (callback) **

Called by: CFSRVT when a vote packet arrives with optional data

This routine is placed in HSHCDA when the access token is formed. The routine is called when the vote packet arrives in order to process the optional data in the packet.

First, HSHBTF is checked to see if this a bit table. If it is, then the structure free count may have to be updated. If structure free count data is present (CFDST1 non-zero) and if the remote node's structure free count transaction number is greater than our own (CFDST1 > CFSSTR(str)), then our copy of CFSSTR is updated and STOFRC is called to update the free count.

Next, we continue at CFADAR and check CFDT1 to see if any file length optional data is present. If it is and the transaction number is greater than ours, we return +2 to CFSRVT and store the data and transaction number in HSHOPT and HSHOP1 and set HSHODA to indicate optional data is present. (This data will be taken out of the resource block and placed in OFNLEN by CFSAWT/CFSAWP). If the transaction number in the packet is not greater than ours, we return to CFSRVT and ignore the optional data.

** CFSFOD - DDMP force out done **

Called by: DDOCFS in PAGUTL

This routine is responsible for signaling that DDMP has completed the force out of OFN pages. The corresponding entry in CFSOFN is checked to see if an access token exists for it. If the table entry is empty, then this node no longer has the resource so SPTST (the OFN access) and SPTFR (the DDMP force out flag bits) are cleared and CFSFOD returns successfully.

If there is a CFSOFN table entry, it contains the address of the resource block for the access token. We retrieve the number of sharers of the resource from HSHCNT and, if greater than 2, we cannot signal success so we return failure. Next, any optional data that is present in the access token (HSHOPT and HSHOP1) is placed in a newly acquired vote packet (obtained via a call to GVOTE1). Note that if HSHODA is zero, then there is no optional data in the token so the file length information is obtained directly from OFNLEN. If we own exclusive access to the OFN, then the transaction number (HSHOP1) is incremented to insure that the remote system will use the optional data we are providing since it is the most current. If this is a bit table OFN, SNDFRC is called to place structure free count optional data into the vote packet.

Next, we clear HSHRFF and set the new access of the OFN. If SPTSR is set, then HSHTYP is set to .HTPLH and SPTST is set to 0. If SPTSR is not set, then HSHTYP becomes .HTOAD and SPTST becomes .SPSRD. Finally, we clear SPTFR, decrement HSHCNT to "unown" the resource and call SCASND to send the vote packet. The packet type code is a -2, which is a "condition satisfied" message which is used to indicate to the remote that the DDMP force out has completed. We will then return successfully.

2. Directory locks and directory allocation - Directory locks are now managed by CFS and are a CFS resource. The old LOKTAB and associated storage is now gone. Each time a directory is locked or unlocked, a CFS resource is created or modified. The remaining directory allocation of each active directory is also a CFS resource.

Below is the table of contents fragment from CFSSRV which illustrates the various directory related routines:

Directory Lock Token

CFS routines:	CFSLDR - Lock directory
	CFSRDR - Unlock directory

Directory locks are the only example of long resource blocks. Also, directory locks are always exclusive (.HTOEX) resources. For the duration of the lock, the process is CSKED.

** CFSLDR - Lock directory **

Called by: LCKDNM in DIRECT (via CALLRET)

First, a check is made to see if the resource already exists. If it does, and it is in "use" (HSHWVT or HSHCNT are set), then a vote is required. Otherwise, we can lock the directory and the following is updated in the resource block:

HSFLAG	HSHCNT incremented (owned)
HSHFRK	FORKX of running fork
HSHTIM	TODCLK stamp to indicate when resource acquired

If the resource is not known to this node, or the current resource block for the directory is in use, then a vote is required. First, CFSSPL is called to obtain a long resource block. CFSSPL will return a block with HSHLOS set in HSFLAG, HSHPST, and HSHOKV zeroed as well as HSHRET set to 1,,LNGADD. The following is then placed in the block:

HSHROT	six-bit structure name
HSHQAL	DRBASE + directory number
HSFLAG	HSHTYP=.HTOEX
HSHCOD	DRBASE
HSHFCT	TODCLK+DIRTIM

Finally, CFSGTT is called to get the token with "retry until successful" set so, upon return, we will have acquired the resource. (Note, if the structure is set local exclusive, CFSGTT will discover this and use CFSGTL. This will mean that HSHVTP will not be updated with the access of the vote since no vote is required.) The following has been updated in the resource block:

HSFLAG	HSHVTP=access of vote, HSHCNT incremented (owned)
HSHFRK	FORKX of running fork
HSHTIM	TODCLK stamp when vote approved and token obtained

If, upon return from CFSGTT, we do not need the newly created block (T1 is non-zero), LNGADD will be called to return the extra block to the CFS pool.

** CFSRDR - Unlock directory **

Called by: ULKDNM in DIRECT (via CALLRET)

The routine CFSRDR simply calls CFSNDO to release the directory lock token. For these long blocks, CFSNDO will do something extra before the call to CFSOHS; it will check the HSHBTT for any waiting forks to wake up. If it finds one, CFSOHS will not be called to notify remotes of the token release. In this way, the forks on the local node will have priority over forks on remote nodes for acquiring directory locks. See the discussion of CFSAWT/CFSAWP for a description of CFSNDO.

Directory Allocation Token

CFS routines:	CFSDAU	- Acquire allocation entry	
	CFAFND/CFAGET	- Find/Get allocation table	
	CFASTO	- Store new allocation value	
	CFAULK	- Unlock allocation entry	
	CFAREM	- Remove allocation entry	
	CFAUPB	- Undo keep here bit	
	CFAVOK	- Vote to be approved	(callback)
	CFADAR	- Optional data present	(callback)
	CFARMV	- Voter remove entry	(callback)

The remaining allocation of a directory is cached in memory to aid in processing page faults and file page creation. For each active directory, there is an allocation entry and this entry is a CFS resource. The optional data is the resource block carries the remaining allocation for this directory.

** CFSDAU - Acquire allocation entry **

Called by: Various routines in PAGEM and PAGUTL. Each calls provides one of the following function codes:

.CFAGT==:0	-	Get and lock current allocation
.CFAST==:1	-	Store allocation
.CFARL==:2	-	Release allocation table
.CFARM==:3	-	Remove entry
.CFAUP==:4	-	Undo hold bit
.CFAFD==:5	-	Find it

This routine is called and a function code is provided. Then, based on this code, we dispatch off to the correct routine.

Function code	Dispatch routine
.CFAGT	CFAGET
.CFAST	CFASTO
.CFARL	CFAULK
.CFARM	CFAREM
.CFAUP	CFAUPB
.CFAFD	CFAFND

** CFAFND/CFAGET - Find/Get allocation table **

Called by:	QLOK	in PAGEM
	ASGALC	in PAGUTL (with CF%PRM set)
	REMALC	in PAGUTL
	GETCAL/GETCAH	in PAGUTL (with CF%NUL and CF%HLD set)

These routines are used to lock the allocation table. CFAFND differs from CFAGET only in that it will not create a new resource block; it will only succeed if the block already exists and can be found via the routine GETDBK. Currently, CFAFND is never dispatched to. CFAGET will return +1 if a resched took place during its operation and +2 if one did not.

Upon entry to CFAGET, the access is determined. If CF%HLD was specified in the flag bits (the left half of T3, the Flags,,Operation word), then the requested access is exclusive (.HTOEX). Otherwise, the access is for full sharing (.HTOAD). GETDBK is called to obtain the address of the resource block. If one does not exist, we continue at CFSDAO.

At CFSDAO, we will create a new resource block and vote for access to the token. CFSSPC is called to get a short request block. CFSSPC will return a block with HSFLAG, HSHPST, and HSHOKV zeroed as well as HSHRET set to 1,,SHTADD. Then, the following is placed in the block:

HSHROT	six-bit structure name
HSHQAL	DRBAS0+directory number
HSFLAG	HSHTYP=access, HSHKPH set if CF%PRM specified
HSHPST	1,,CFARMV
HSHOKV	1,,CFAVOK
HSHCDA	1,,CFADAR
HSHOPT	0
HSHOP1	0 (transaction number)

Finally, CFSGTT is called to get the token (with "retry until successful" set). (Note, if the structure is set local exclusive, CFSGTT will discover this and use CFSGTL. This will mean that HSHVTP will not be updated with the access of the vote since no vote is required.) When the token is acquired, the following has been updated in the resource block:

HSFLAG	HSHVTP=access of vote, HSHCNT incremented (owned)
HSHFRK	FORKX of running fork
HSHTIM	TODCLK stamp when vote approved and token obtained

If, upon return from CFSGTT, we do not need the newly created block (T1 is non-zero), SHTADD will be called to return the extra block to the CFS pool.

Next, HSHLOK is called to retrieve the resource block and the access desired in the call to CFSDAU is checked. If it is not CF%HLD, then we do not want write access so we "unown" the resource by decrementing HSHCNT.

Finally, HSHOP1 was checked to see if optional data was returned in the voting process. If it is non-zero, then there was optional data returned (the current allocation) and it is in HSHOPT. We return +1 from CFAGET (and CFSDAU) now with T1 containing the current allocation, T2 the resource block address, and T3 the transaction number. If HSHOP1 was zero, then no optional data was returned. This means that no other node had more recent information that us to contribute so we must establish ourselves as the node that holds the latest information. The value of the allocation (passed into CFSDAU in T3 and held in ALLC) is placed in HSHOPT and HSHOP1 (the transaction number) is incremented only if this is not a temporary entry (CF%NUL was not specified in the flag bits passed into CFSDAU). We return +1 with the allocation in T1 and resource block address in T2.

The description above outlines what happens when CFAGET is invoked to return the allocation for a resource which did not exist before on this node. However, if one already exists (and is found by GETDBK) then the following takes place.

First, we check the access bits set in T2 to see if this is a permanent block (CF%PRM set) and, if so, HSHKPH is set in the HSFLAG word. A check is then made to see if anyone is using this block by checking HSHTWF, HSHWVT, and HSHUGD. If any of these bits are set, then we wait a while at routine CFGVOT. This wait routine will wakeup when HSHTWF, HSHWVT, and HSHUGD are all zero. Upon wakeup, we try again at the top of the CFAGET routine.

If no one is waiting for this block, we check to see what kind of access we have to this resource (in HSHTYP). If it matches the type of access we are requesting, HSHCNT is incremented. Otherwise, our access must be upgraded. We set HSHTWF to indicate we are waiting for this token, and call CFSUGD to upgrade our access. If CFSUGD allows the upgrade, we will have obtained the desired access to the resource and HSHTWF will be cleared. Otherwise, CFSUGD has denied our upgrade attempt and we must wait. If HSHWTM is non-zero (the wait time for when to try again) then this is placed in HSHTIM. If HSHWTM is zero, then TODCLK+^D500 is placed in HSHTIM. Then we wait at CFSRWT. Upon return from CFSRWT (see the discussion of file access token for the conditions of return for CFSRWT), we try to get the token again by continuing at the top of CFAGET.

Once we have acquired the desired access to the token, we check to see if we wanted write access (CF%HLD set). If not, HSHCNT is decremented to "unown" the resource. Next, our fork number is placed in HSHFRK. Finally, T1 is loaded with the allocation (HSHOPT), T3 with the transaction number (HSHOP1), and T2 with the resource block address. We will return +1 if we ever had to enter a wait routine or if we needed to upgrade our access. Otherwise, we will return +2.

** CFASTO - Store new allocation value **

Called by:	QSET	in PAGEM
	ADJALC	in PAGUTL

This routine is used to place a new allocation value into the resource block for particular directory allocation token. Once the value has been stored, the resource will be released. So, a store a an allocation entry has an implied release following.

The block is located via a call to GETDBK. Then the allocation value (which is passed in and resides in ALLC) is placed in HSHOPT. The transaction number is incremented (HSHOP1) to insure that this is the most current entry known. Finally, we fall through into CFAULK to release the resource. A description of CFAULK follows.

** CFAULK - Unlock allocation entry **

Called by:	QREL	in PAGEM
	GETCAL/GETCAH	in PAGUTL

The routine CFAULK simply calls CFSNDO to release the directory allocation token. See the discussion of CFSAWT/CFSAWP for a description of CFSNDO.

** CFAREM - Remove allocation entry **

Called by:	REMALC	in PAGUTL
	GETCAL/GETCAH	in PAGUTL

This routine removes a resource from the hash table. The resource block is located via a call to GETDBK and HSHCNT is decremented. If the number of sharers goes to zero, then no one is using this resource and CFSRMV is called to removed it from the hash table. CFSRMV will not post the removal.

** CFAUPB - Undo keep here bit **

Called by: DASALC in PAGUTL

This routine is used to "unlock" a resource from the node. The resource block is found via a call to GETDBK and the "keep here" bit (HSHKPH) is cleared. (This will allow the routine CFARMV to signal to CFSRMX that this resource block should not be held on the node and can be removed from the hash table. CFSRMX is called from CFSRTV when an incoming vote results in releasing the resource. If HSHKPH is set when CFARMV is called, the resource becomes a place-holder since it was desired that the block always remain on this node.) Finally, CFSNDO is called to undeclare the resource. See the discussion of CFSAWT/CFSAWP for a description of CFSNDO.

** CFAVOK - Vote to be approved (callback) **

Called by: CFSRTV when vote is to be approved

This routine is placed in HSHOKV when the allocation token is formed. The routine is called when a vote is to be approved for this resource in order to place this node's optional data (the allocation data held by this node) in the vote packet. The "optional data present in vote" flag (CFODA) is set in the vote packet so that the optional data will be noticed by CFSRVT when this vote packet is received.

** CFADAR - Optional data present (callback) **

Called by: CFSRVT when a vote packet arrives with optional data

This routine is placed in HSHCDA when the allocation token is formed. The routine is called when the vote packet arrives in order to process the optional data in the packet.

The transaction number present in the packet is compared to our own in HSHOP1. If the packet's value is greater than ours, we return +2 to CFSRVT and store the data and transaction number in HSHOPT and HSHOP1 and set HSHODA to indicate optional data is present. (This will be noticed by CFAGET and the allocation will be returned to the caller.) If the transaction number in the packet is not greater than ours, we return to CFSRVT and ignore the optional data.

** CFARMV - Voter remove entry (callback) **

Called by: CFSRTV when we want to release the resource
(Note: CFSRSE has the ability to call a post routine but it should never be called for an allocation token since HSHKPH will be set and this will prevent the block's removal.)

This routine is the callback routine placed in HSHPST when the allocation token is formed. To post removals, CFSRMX is called. CFSRMX is the alternate entry point to CFSRMV used to do posting of removals. CFSRMX will insure that CFARMV is called to do any cleanup that is needed before the resource block is removed from the hash table.

Basically, CFARMV checks HSHKPH to see if the block should be kept on the node. If not, CFARMV will return +1 and CFSRMX will remove the resource block from the hash table and return indicating the resource is unconditionally available. Otherwise, the block is to be kept on the system so HSHTYP is zeroed (this sets the state to place-holder; .HTPLH) and return +2 to CFSRMX. CFSRMX will then zero HSHCNT and HSHTYP and return to CFSRTV with 0 in T1. This value will be used as the vote type (.CFTYP) to the other node; 0 indicates unconditional yes. (Note that CFARMV assumes T1 contains the address of the vote packet and is, therefore, not 0. So, before returning to CFSRMX, T1 is not changed. This is important because CFSRMX will check T1 and, if it is zero, assumes that this is a delayed yes return for a file access token. So, CFARMV is depending upon the fact that T1 is non-zero when returning to CFSRMX. This should never change.)

3. Structures - Structure mounting is managed by CFS in order to coordinate access to the structure by 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 the accessing processors. This is accomplished by providing 2 resources to control structures: structure name and drive serial number resources.

Below is the table of contents fragment from CFSSRV which illustrates the various structure related routines:

Structure Name Token and Drive Serial Number Token

CFS routines:	CFSSMT - Acquire structure resource
	CFSSUG - Upgrade or downgrade mount
	CFSSDM - Release mount resource
	CFMNAM - Register structure name
	CFMDSN - Register drive serial number

In order to mount a structure, both the structure name token and drive serial number token must be acquired. The access type (whether the structure is accessed shared or exclusive) is controlled by the DSN resource only. The structure name token is always created with full sharing. When a structure's access type is changed, only the DSN resource needs to be updated. 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 CFS at CFRDSN describing the old and new UDB for the disk pack.

** CFSSMT - Acquire structure resource **

Called by:	MNTPS in DSKALC	(for exclusive and shared)
	MSTMNT in MSTR	(access based on user flags)

This routine is called when a structure is first mounted on a system. It, in turn calls routines CFMNAM to register the structure name and CFMDSN to register the driver serial number. CFSSMT insures that the alias for the structure is not already in use for another structure and that the name of the structure is the same as what is in use by any other CFS system. These two conditions are sufficient to allow the alias to be used as the root name of the structure.

Upon entry to CFSSMT, a check is made to see if this is a "reduced" CFS system. This is controlled by defining the switch CFSDUM. If this is defined, then this is a "reduced" CFS. This system is on the CI and uses SCA to connect to other CI-based systems. However, this processor will not share structures with any other system but will insure that the structures it is using are mutually exclusive from structures used by any other CI-based TOPS-20 system. This implies that this system will establish connections to other reduced or full CFS systems and will participate in structure mounting votes.

If this is a reduced system (MYPOR1 is not less than zero), then the access is forced to be exclusive (.HTOEX). Otherwise, the access is determined from the call (passed in T2) and it will be set to either full sharing (.HTOAD) or exclusive (T2 is zero = .HTOAD, otherwise .HTOEX).

Finally, CFMNAM is called to register the name and CFMDSN is called to register the serial number. If the call to CFMNAM fails, we RETBAD and if CFMDSN fails, we continue at CFSSDM to undo the mount and then return failure.

At CFMNAM, we will create a new resource block and vote for access to the structure name token. CFSSPC is called to get a short request block. CFSSPC will return a block with HSFLAG, HSHPST, and HSHOKV zeroed as well as HSHRET set to 1,,SHTADD. Then, the following is placed in the block:

HSHROT	six-bit structure name
HSHQAL	STRCTN
HSFLAG	HSHTYP=.HTOAD, HSHAVT set, HSVUC set
HSHCOD	UDBDSN XOR (STRCTK+UDBDSH)

Finally, CFSGET is called to get the token (with "try only once" set). If the token is acquired, the following has been updated in the resource block:

HSFLAG	HSHVTP=access of vote, HSHCNT incremented (owned)
HSHFRK	FORKX of running fork
HSHTIM	TODCLK stamp when vote approved and token obtained

If, upon return from CFSGET, we do not need the newly created block (T1 is non-zero), SHTADD will be called to return the extra block to the CFS pool.

At CFMDSN, we will create a new resource block and vote for access to the drive serial number token. CFSSPC is called to get a short request block. CFSSPC will return a block with HSFLAG, HSHPST, and HSHOKV zeroed as well as HSHRET set to 1,,SHTADD. Then, the following is placed in the block:

HSHROT	UDBDSN
HSHQAL	STRCTK+UDBDSH
HSFLAG	HSHTYP=access, HSHAVT set, HSVUC set
HSHCOD	six-bit alias name

Finally, CFSGET is called to get the token (with "try only once" set). If the token is acquired, the following has been updated in the resource block:

HSFLAG	HSHVTP=access of vote, HSHCNT incremented (owned)
HSHFRK	FORKX of running fork
HSHTIM	TODCLK stamp when vote approved and token obtained

If, upon return from CFSGET, we do not need the newly created block (T1 is non-zero), SHTADD will be called to return the extra block to the CFS pool.

If we just got the token exclusively, the STEXL flag is set in the status bits of the SDB for that structure.

** CFSSUG - Upgrade or downgrade mount **

Called by: MNTCSM in MSTR

This routine is used to change the access to a mount resource. It is only valid for full CFS systems. The new access is passed in T2 (T2 is zero = .HTOAD, otherwise .HTOEX). Then the resource is located via a call to HSHLOK an CFSUGA is called to upgrade (or downgrade) to the desired access.

** CFSSDM - Release mount resource **

Called by: MNTER4 in MSTR
MSTDIS in MSTR

This routine is called to release the mount resource upon a dismount. For both the resource name token and the drive serial number token, the resource block is found via a call to HSHLOK and then removed via a call to CFSRMV. CFSRMV will not post the removal.

4. BAT Block Lock - The BAT block lock on a structure is now a CFS resource. It is an exclusive resource.

Below is the table of contents fragment from CFSSRV which illustrates the various BAT block related routines:

BAT Block Lock Token

:FS routines:	CFGBBS - Set BAT block lock
	CFFBBS - Release BAT block lock

** CFGBBS - Set BAT block lock **

Called by: LKBAT in DSKALC

Upon entry to CFGBBS, CFSSPC is called to obtain a short resource block. CFSSPC will return a block with HSFLAG, HSHPST, and HSHOKV zeroed as well as HSHRET set to 1,,SHTADD. The following is then placed in the block:

HSHROT	six-bit structure name
HSHQAL	-1
HSFLAG	HSHTYP=.HTOEX

Next, a call is made to get the resource. CFSGET is called if the structure is shared. If the structure is set exclusive, CFSGTL is used. (Note that if CFSGTL is called, HSHVTP will not be updated with the access of the vote since no vote is required.) The call is made to "retry until successful" so, upon the return, we have acquired the resource. The following has been updated in the resource block:

HSFLAG	HSHVTP=access of vote, HSHCNT incremented (owned)
HSHFRK	FORKX of running fork
HSHTIM	TODCLK stamp when vote approved and token obtained

If, upon return from CFSGET, we do not need the newly created block (T1 is non-zero), SHTADD will be called to return the extra block to the CFS pool.

** CFFBBS - Release BAT block lock **

Called by: ULKBAT in DSKALC

The routine CFFBBS simply calls CFSNDO to release the BAT block lock token. See the discussion of CFSAWT/CFSAWP for a description of CFSNDO.

5. ENQ/DEQ - In order to allow ENQ/DEQ to operate in a CFS environment, there is a temporary CFS resource representing the ENQ on a file. Note that this is not the same as global ENQ/DEQ across all the systems in a CFS environment. This is always an exclusive resource.

Below is the table of contents fragment from CFSSRV which illustrates the various ENQ related routines:

ENQ Token

CFS routines:	CFSENQ - Get ENQ resource
	CFSDEQ - Release ENQ resource

Each time an ENQ file resource is first requested (an ENQ lock block is created), CFS is called to register an exclusive ENQ resource for the file. If the requesting processor succeeds in creating the ENQ resource, then the ENQ will be allowed. Otherwise, the ENQ is denied.

** CFSENQ - Get ENQ resource **

Called by: CFETST in ENQ

Upon entry to CFSENQ, ENQSET is called to set up T1 and T2 with the proper root and qualifier. Then CFSSPC is called to obtain a short resource block. CFSSPC will return a block with HSFLAG, HSHPST, and HSHOKV zeroed as well as HSHRET set to 1,,SHTADD. The following is then placed in the block:

HSHROT	six-bit structure name
HSHQAL	FILEEQ+index block address
HSFLAG	HSHTYP=.HTOEX, HSHLCL is set

Next, a call is made to get the resource. CFSGET is called if the structure is shared. If the structure is set exclusive, CFSGTL is used. (Note that if CFSGTL is called, HSHVTP will not be updated with the access of the vote since no vote is required.) The call is made to "try only once". If we acquire the resource, the following has been updated in the resource block:

HSFLAG	HSHVTP=access of vote, HSHCNT incremented (owned)
HSHFRK	FORKX of running fork
HSHTIM	TODCLK stamp when vote approved and token obtained

If, upon return from CFSGET, we do not need the newly created block (T1 is non-zero), SHTADD will be called to return the extra block to the CFS pool.

** CFSDEQ - Release ENQ resource **

Called by:	CRELOK in ENQ
	LOKREL in ENQ

The routine CFSDEQ simply calls CFSNDO to release the ENQ token. See the discussion of CFSAWT/CFSAWP for a description of CFSNDO.

Appendix A