SCA Design Notes

Judy Hall
Dave Lomartire

This is a collection of notes about the changes that have been made to SCAMPI. It is not intended to be a complete description of the changes, nor is it in any way a tutorial description of the design. But it does document the reasons behind some decisions, and may provide some insight into the assumptions implicit in the code.

Interactions in SCAMPI
Judy Hall
6/14/84

1. System block's list of connection blocks (pointed to by .SBFCB)

NOTE: We assume that the c.b. lock has prevented setting the "reap" bit until no other code wants to use the c.b. The concern here is only with keeping the list intact.

Adding - SC.LCB -- CIOFF

Removing -- SC.RCB -- CIOFF

Traversing -- SC.RAP knows it's the only deleter, and uses no protection when traversing
-- SC.SCM knows it's at interrupt level, uses no protection when traversing
-- SC.IDL goes NOSKED when traversing to prevent SC.RAP from running

Nothing totally eliminates the queue. It may become empty when the last entry is removed.

2. Don't-care listeners (pointed to by TOPDC)

Adding -- SC.LCB -- CIOFF

Removing -- SC.RCB -- CIOFF
-- SC.SCM -- interrupt level

Traversing -- SC.RAP uses CIOFF. WHen it finds an entry to be reaped, it goes CION. When it continues, it returns to the top of the list.

3. System block's work queue (pointed to by .SBTWQ)

Adding -- SC.AWQ (from SC.SCA, SC.SNM, SC.DEF) -- CIOFF, checks v.c. before returning entry. If closed, doesn't queue the entry since SC.ERR has destroyed the queue

Deleting -- SC.RWQ (from SC.SNM, SC.DEF) -- CIOFF

Traversing -- SC.SNM, SC.DEF -- always remove entry while CIOFF, return it while CIOFF

Destroying queue -- SC.ERR -- CIOFF when resetting the head and tail

At any time, SC.ERR may destroy the list. Others can't traverse it CION, or add an entry without verifying that node is still online.

4. System block's address of outbound buffer (pointed to by .SBOBB)

Taking the buffer -- SC.SNM -- uses EXCH with zero; if result is non-zero, owns the buffer

Returning buffer to s.b. -- SC.SNM -- while CIOFF, makes sure v.c. is open. If open, stores address in s.b.; if closed, returns buffer to port queue
-- SC.SAR -- knows it's at interrupt level, stores address in s.b.

Returning buffer to port or SCA --

SC.ERR -- uses EXCH with zero; if result is non-zero, returns buffer to SCA's pool
-- if zero, gets a buffer from the port's queue.

SC.RIB -- knows that v.c. is already closed, returns buffer to port

SC.SNM -- if PHYKLP refuses to send, returns buffer to port

At any time, SC.ERR may return the buffer to SCA's pool. All code must test and zero the address in one operation.

5. Connection block

Connection block lock protects against

A. Reaping

Makes c.b. address invalid
Manages buffers according to counts in c.b.
(Assume reaping can happen as soon as unlock and OKSKED)

B. Protocol violation

Disc_req / application message
Disc_req / disc_req
Disc_req / credit_req

(Even if last is not a violation of protocol, we will have reaped the block by the time the response arrives. And there is no reason to change credit if we are disconnecting.)

C. Change in connection state or block state

Protocol is generally synchronous, but

1. SC.DIS at odd times can be interrupted by incoming protocol message. Both change state.

2. Sysap can do connection management request, interrupt out of it, and do SC.DIS.

3. Any connection management request can be interrupted by node offline.

Events that can't tolerate these changes --anything that believes the c.b.a. is valid, sets the connection state, sets the block state, or sends a packet, including

- Calls from the sysap, including
  - Connection management requests
  - Sending messages and datagrams
  - Queueing or canceling buffers for reception
  - Sending DMA
- Idle chatter
- Sending connection management requests

These all lock the connection block lock. In the case of sending a packet, they unlock it only after the packet has been given to PHYKLP.

It is NOT necessary to leave a c.b. locked while it is waiting to send a connection management request (that is, the buffer is in use). However, it must be locked while its request is being sent and its state is being changed.

Some code needs to honor the lock if it's at interrupt level, and lock the lock if it's not. So there are three kinds of routines:

A. Lock

> If at interrupt level, do nothing
> If not, AOS the count.

B. Honor

> Called by code that runs only at interrupt level
> If lock is unlocked, proceed.
> If lock is locked, set a bit to indicate what is deferred, and don't proceed.

C. Lock and honor

> If not at interrupt level, do "lock"
> If at interrupt level, do "honor"

When locking, code goes NOSKED. Therefore, there can be only one locker at a time.

Events that defer:

- SC.ERR - node offline
- SC.DIS - sysap requests disconnect
- SC.DRQ - remote side initiates disconnect
- SC.SNM - sending a connection management request

Unlock routine processes according to the bits it finds set.

6. Existence of connection block

Only SC.RAP deletes a connection block. Code that sets the "reap" bit must lock the connection block, which makes it NOSKED. The reaper won't run until the block has been unlocked.

7. Virtual circuit's state (indicated by SBVCST)

```
        SC.ERR -- if no c.b. is locked, calls OPENVC
               -- if a c.b. is locked, increments s.b.'s count of locked c.b.'s

        SC.ULK -- if node offline had been deferred, decrements s.b.'s count
                        of locked c.b.'s. If result is zero, calls OPENVC
```

8. Send credit (indicated by .CBSCD)

Single-instruction AOS or SOS avoids races

Decrement and test -- SC.SMG, SC.SND, SC.REQ -- SOS and load AC

```
Increase -- SC.DMA increases by 1 -- AOS -- interrupt level
         -- SC.CRQ and SC.AMG -- ADDM from packet -- interrupt level
```

Use -- SC.CRQ compares with minimum send credit -- interrupt level

Store initial value -- SC.ORQ and SC.ARQ -- interrupt level

9. Receive credit

CIOFF protects all changes; multiple kinds of credit must be adjusted without interruption.

Use of return_credit prevents interaction between cancel receive message and other events. Avoids cases in which the number of buffers queued differs from receive credit.

Interlock word, .CBPND, prevents conflicts between two contexts that want to send a credit_request. Prevents an attempt at queueing the c.b. a second time, and avoids sending a null credit_request. When the credit_response arrives, we send another credit_request if the need has arisen since the last one was sent.

```
SC.CDT (from SC.RMG, SC.CRM, SC.IDL, SC.CRS) -- EXCH of .CBPND with -1.
                If result is 0, send credit_request. If not, return.
```

SC.CRS -- clear .CBPND and call SC.CDT to send again if necessary.

10. Count of buffers queued for receiving datagrams

Decrement -- SC.CRD -- SOSL once per buffer; AOS if negative

Change -- SC.RDG -- ADDM
       -- SC.ADG -- SOSGE; AOS and drop datagram if negative

Use -- SC.RCB dequeues based on this value; block isn't being used by then

Why SC.DIS is Special
6/21/84

SC.DIS uses CIOFF to lock out SC.DRQ, SC.ORQ, SC.ARQ, SC.ARS. If these were allowed to run, they would have to honor the lock. SC.DRQ already does honor it, but the other three do not.

The alternative to CIOFF is to make each of these honor the lock. The race arises only when the sysap has done a disconnect at an unusual time, as follows:

SC.ORQ - sysap disconnects a listener, and connect_request arrives. SC.DIS sets new state to closed, but SC.ORQ has already set it to connect_received, and sent a connect_rsp Match. Its deferred code would have to manufacture a reject_request if the state had changed to closed. Today it sends connect_rsp NM if the state is closed.

SC.ARQ - sysap disconnects after initiating a connection, and an accept_request arrives. SC.DIS sets new state to closed, but SC.ARQ has already set it to open, and sent an accept_response Match. Its deferred code would have to manufacture a disconnect_request if the state had changed to closed. Today it sends an accept_response NM if the state is closed.

SC.ARS - sysap disconnects after accepting a connection for a listener, and an accept_response Match arrives. SC.DIS sets new state to closed, but SC.ARS has already set it to open. Its deferred code would have to send a disconnect_req, which it does today if the state is closed.

It seems more practical to let SC.DIS be CIOFF until it is ready to send a message (if that's necessary), and avoid a lot of special-purpose code. It already needs to be CIOFF for a large fraction of the execution time anyway. Also, the speed of disconnect doesn't seem terribly critical to the overall performance of the system.

Since SC.DIS is CIOFF for this purpose, it need not lock the lock. SC.DRQ and SC.ERR will be held off by CIOFF, so the lock is unnecessary.

Why Reap in Process Context
Judy Hall
6/14/84

1. Allows the return of buffers that the sysap (CFS) has queued for output. These will be returned at SC.INT, which assumes that the CID is still valid. That is the only way to identify the sysap to which the buffer is to be returned.

2. Code that sets the "reap" bit must still unlock the connection block before the reaper can run. Locking goes NOSKED, which prevents reaping. If we were to reap immediately, each pass through unlock would have to check for this case.

3. Reaping isn't very urgent, and shouldn't be done at interrupt level.

Why Stock the Message Free Queue

Judy Hall

6/20/84

1. When a node goes offline, the outgoing connection management buffer may be i use. If so, we try to get a buffer from the port's queue to give back to SCAMPI. When PHYKLP queues the buffer to the port, the port will return it with error. PHYKLP will return the buffer to the free queue. Thus there is a window where the port will be short one buffer.

2. When a node goes offline, the incoming connection management buffer may be on the response queue. SCAMPI will try to remove a buffer from the port's queue at SC.ERR. Eventually the connection management buffer will be given to SCAMPI, which will return it to the free queue. Meanwhile, there is a window where the port will be short one buffer.

3. When a node goes offline, a sysap may have queued a buffer to go out, without asking that the buffer be returned. In this case, the sysap's receive credit includes this buffer. If the reaper runs before the port sees the buffer, SC.RCB will try to remove a buffer from the port when it hasn't been queued there yet. As soon as the port rejects the command, PHYKLP will give the buffer back to the port.

Incoming Packets on Closed V.C.
Judy Hall
6/22/84

An incoming packet may be on the response queue behind a packet that will cause the v.c. to be closed. SCAMPI will receive a callback at SC.ERR for the bad packet, followed by a callback at SC.INT for the good one.

SCAMPI used to process this second packet normally, even though the v.c. was closed. This led to the following problem:

Suppose the second packet is a connect_request. SCAMPI matched the request with a waiting listener and set the connection state to "connect received". However, the attempt at sending a connect_response failed because the v.c. was closed. The sysap was never notified of the change of state. It believed it had an outstanding listener.

SCAMPI now checks the state of the v.c. for every incoming packet. Because connect_request isn't associated with a particular connection, it's not sufficient to check the connection state. Also, handling of the buffer is different if we know that SC.ERR has already run. This could be via a separate dispatch table at SC.INT, indexed by op code and handling only the case of incoming packet on a closed v.c. Presently, the check on the v.c. is made after the op code is known.

The buffer is handled as follows:

1. Connection management request

This is the buffer that SCAMPI queued to the port when the node came online. Normally, the response goes out in this buffer. If the node is offline, SCAMPI queues the buffer to the port. SC.ERR has already taken a buffer from the port and returned it to its pool. THus this buffer belongs to the port.

2. Connection management response

This is the buffer that is queued to the system block for outgoing requests. When SC.ERR ran, the pointer was 0, and SCAMPI took a buffer from the port's queue. Thus SCAMPI now returns this buffer to the port.

3. Application message

SCAMPI gives the buffer to the port. It was included in the sysap's receive credit, so if the reaper has run, it has already removed an extra buffer from the port's queue.

4. Application datagram

SCAMPI gives the buffer to the port. It was included in the sysap's count of queued datagram buffers, so if the reaper has run, it has already removed an extra buffer from the port's queue.

The List of Connection Blocks

Judy Hall

6/20/84

It's not clear that linking the connection blocks to the system block is essential. The list is useful for idle chatter, for the reaper, and for finding the connections when a node goes offline.

Presently, a c.b. remains on the list until the reaper deletes it. This means that searches through the list will stumble over this block. This is most likely to be visible after a node goes offline. All its connection blocks are marked to be reaped, and a whole new set of them is created, but they are behind the obsolete ones.

One possibility seems to be to remove a c.b. from this list at SC.ERR, or at SC.PTC, which is reached also when a disconnect sequence completes. These blocks could be on a special "reap" linked list, which the reaper would search instead of searching the list for each system block. Since the reaper runs periodically, and typically searches these lists without accomplishing anything, this would seem to offer a performance gain.

Other possibilities:

Set a bit in CIDTAB to indicate that the connection block needs reaping, and have the reaper scan CIDTAB.

Set a global flag when the reap bit is set in any c.b., and don't call the reaper unless this flag is set.

Opening a V.C.
Judy Hall
6/20/84

SC.SNM may try to queue a packet after SC.ERR has run. Between getting the buffer (.SBOBB) and sending, it doesn't check the v.c. We need to be sure that 1) PHYKLP won't accept the packet if the v.c. is closed, and 2) the v.c. doesn't get reopened before we queue the packet.

Sequence in PHYKLP should always be:

1. Mark the state as closed
2. Tell the port to close the v.c.
3. Tell SCAMPI the v.c. is closed

SCAMPI should never allow the v.c. to be opened as long as any connection block is locked. The locker may try to send a packet (SC.SNM, SC.SMG, etc.), and we want that to fail.

Closing and Opening V.C.'s (SCAMPI and PHYKLP)
Judy Hall
6/10/84

1. SCA makes the decision to close

   SCA calls PHYKLP, which
   a. marks the state as "closed"
   b. tells the port to close the v.c.
   c. calls SC.ERR

2. PHYKLP makes the decision to close

   It follows the same sequence as in #1 above.

3. The port informs PHYKLP of an error

   PHYKLP either does all of #1 above, or leaves out telling the port to close the v.c.

If PHYKLP is unable to get a buffer for the SETCKT, it sets a bit in RIDSTS (or elsewhere) indicating that the v.c. still needs to be closed. At this point, the system block has the state as closed, and SENDVC is refusing to queue packets to the node.

The poller checks this bit, and sends the SETCKT whenever it can.

- - - - -

SCA decides it can open a v.c. (PHYKLP never decides that if the v.c. has ever been open before).

SCA calls PHYKLP, which sends a start unless the "need to close" bit is still on. In that case, or if no buffer is available, it sets a bit in RIDSTS called "need to open".

The poller always checks "need to close" before it checks "need to open". When it wants to open, it sends a start.

- - - - -

Neither routine returns failure. The purpose of SCA's "open" call is to say "you can open whenever you want to". PHYKLP will notify SCA when the v.c. has been opened, and SCA is not concerned with the events that prevent this.

Similarly, when SCA says "close the v.c.", it means "don't queue any more of my packets". So PHYKLP should mark its data base and act as if the v.c. is closed, even if it hasn't found a way to tell the port yet.

Multiple attempts at closing, either by SCA or because of the arrival of bad packets, should be filtered. SCAMPI checks for multiple calls to SC.ERR for the same system block, but there seems to be no reason to perform the close function more than once.

- - - - -

When queueing a START for "open v.c.", PHYKLP should use the lowest possible priority. THus, if any message has been queued to the port, the port will reject it before sending the START. Otherwise, stale data might go out on the newly-opened v.c.

Handling of Buffers on Error
Judy Hall
6/17/84

I believe that PHYKLP should return both message buffers and datagram buffers to the port's queues when these buffers are locally-generated, the "response" bit is not on, and an error has occurred. Here is my reasoning:

1. Message buffers

A. Those sent by a sysap.

If the sysap doesn't want the buffer back, SCAMPI increments its receive credit on the assumption that the buffer will be put on the free queue when the port has finished with it.

Indeed, if there is no error, the port puts the buffer on the free queue, where it is available for an incoming message. If there is an error, the count has still been incremented, so the buffer should ultimately wind up on the free queue.

If the node goes offline after the sysap has sent a message, in theory the reaper could run before the port decides to refuse to send the message. In this case, SCAMPI would remove "n" buffers from the port's queue, where "n" includes the 1 that was added to it when the message was sent. The buffer, then, belongs to the port.

B. Those sent by SCAMPI

SCAMPI sends its requests in a buffer that has been allocated especially for the purpose (one buffer per remote node). It assumes that the remote system will send a response, for which the port will acquire a buffer from the free queue. Thus it does not set the "response" bit when it sends a request.

If the node goes offline, and a request is outstanding, the port will return the packet with error. Meanwhile, SCAMPI will have been called at SC.ERR, where it will determine that the buffer is in use, and move a buffer from the free queue to SCA's pool. When the original buffer is finally available, it should replenish the free queue.

2. Datagrams

As with messages, the sysap may queue a datagram buffer without asking for a response. In that case, SCAMPI counts it as being queued to the port for an incoming datagram. (This accounting is similar to receive credit for messages, but it is not communicated to the other host.)

If no error occurs, the port returns this datagram buffer to the port's free queue. Therefore, if there is an error, PHYKLP should return such a buffer to the port's queue.

SCA Buffer Management
David Lomartire
14-Sep-84

o SC.ALC - Allocate message and datagram buffers for SCA pool

Called by: SC.ALM via job 0 when DDCFSF is non-zero

Calls: SC.CMG to create message buffers
SC.CDG to create datagram buffers

This routine is responsible for keeping the SCA pool of datagrams and messages up to the minimum thresholds. Whenever a buffer request is made which causes the buffer count (FQCNT or DFQCNT) to fall below the minimum, DDCFSF is set. This will make job 0 call SC.ALM which will call SC.ALC.

For messages -
The number of buffers in FQCNT is checked against the minimum threshold (MINMSG). If it is below, a call to SC.CMG is made to create more message buffers to bring us up to the minimum threshold. Upon successful return from SC.CMG, the buffers are linked to BOTFQ, FQCNT is updated, and TOTMGB (total number of buffers created so far) is updated.

For datagrams -
The number of buffers in DFQCNT is checked against the minimum threshold (MINDG). If it is below, a call to SC.CDG is made to create more datagram buffers to bring us up to the minimum threshold. Upon successful return from SC.CDG, the buffers are linked to BOTDFQ, DFQCNT is updated, and TOTDGB (total number of buffers created so far) is updated.

o SC.DEF - Handle buffer deferral

Called by: SC.ALM via job 0 when DDCFSF is non-zero

Calls: SC.BF2 to get (and create if necessary) buffers

This routine handles buffer deferral. When a connect block is stuck on buffers, the bit corresponding to the node number of its system block is set in SBSTUK and DDCFSF is set. The connect block is queued to the system block work queue. When job 0 runs, SC.DEF will run and call SC.BF2 in an attempt to get the buffers it needs to send the connect_request or accept_request.

When run, SC.DEF "scans" SBSTUK to find the first system block with any connect blocks which are stuck on buffers. The stuck bit for this system block is cleared in SBSTUK and all the connect blocks which are stuck are processed. Any connect block that became unstuck are placed on the end of the work queue. Once all connect blocks are done, a call is made to SC.SNM in an attempt to send out the queued connection management requests for that system block.

All system blocks which are stuck are processed until SBSTUK is zero (indicating that there are no longer any stuck connect blocks).

o SC.BUF (SC.BF1 - Allocate buffers from pool; cannot create buffers)
(SC.BF2 - Allocate buffers from pool; can create buffers)

SC.BF1 called:	SC.SNM to get initial buffers for c.b.
SC.BF2 called:	SC.DEF when processing stuck connect blocks
Calls:	SC.ABF to allocate message buffers
	SC.ALD to allocate datagram buffers
	SC.CMG to create message buffers
	SC.CDG to create datagram buffers
	LNKMFQ to link message buffers to port
	LNKDFQ to link datagram buffers to port
	SC.RBF to return extra message buffers from SC.CMG
	SC.RLD to return extra datagram buffers from SC.CDG

These routines are called by SC.SNM (SC.BF1) and SC.DEF (SC.BF2) to allocate buffers for a connect_request or accept_request from the SCA pool.

For messages -
The number of buffers needed in CBIMB is checked to see if any are required. If so, a call to SC.ABF is made to obtain them.

If this fails, (and if buffers cannot be created) DMRCNT is incremented to indicate this and a failure return results. If buffers can be created a call is made to SC.CMG to create the needed buffers. A failure here results in a failure return from SC.BUF. On success, the total number of buffers created is added to TOTMGB. A check is made to see if more were created than were required and, if so, the extras are returned to the SCA pool via a call to SCM.RM (which calls SC.RBF once for each extra buffer).

Once the buffers have been obtained, a call is made to SCL.MC to link each message buffer onto the port message free queue (via call to LNKMFQ). Pending receive credit (.CBPRC) is updated to reflect these buffers. Finally, the number of buffers to queue (CBIMB) is set to zero.

For datagrams -
The number of buffers needed in CBIDB is checked to see if any are required. If so, a call to SC.ALD is made to obtain them.

If this fails, (and if buffers cannot be created) DDRCNT is incremented to indicate this and a failure return results. If buffers can be created a call is made to SC.CDG to create the needed buffers. A failure here results in a failure return from SC.BUF. On success, the total number of buffers created is added to TOTDGB. A check is made to see if more were created than were required and, if so, the extras are returned to the SCA pool via a call to SCM.RD (which calls SC.RLD once for each extra buffer).

Once the buffers have been obtained, a call is made to SCL.DC to link each datagram buffer onto the port datagram free queue (via call to LNKDFQ). The count of datagram buffers queued (.CBDGR) is changed to reflect these buffers. Finally, the number of buffers to queue (CBIDB) is set to zero.

o SC.SBT - Set buffer thresholds

Called by: SCA upon initialization to establish starting levels
SC.ONL to update levels when node comes online
SC.ERR to update levels when node goes offline

This routine is used to set the value for the minimum level of message (MINMSG) and datagram buffers (MINDG) to be maintained by SCA. This count is based on the number of systems currently online (SBCNT). Whenever a request for a buffer forces the count of buffers to fall below the minimum, DDCFSF is set to signal job 0 to run SC.ALC and allocate more buffers for the SCA pool.

MINMSG = (SBCNT*MBPS)+MGTRSH, where MBPS is the number of message buffers to maintain per online system and MGTRSH is the base value of buffers which must be available.

MINDG = (SBCNT*DBPS)+DGTRSH, where DBPS is the number of datagram buffers to maintain per online system and DGTRSH is the base value of buffers which must be available.

o SC.ABF - Allocate message buffers from SCA pool

Called by: SCA to acquire a buffer for NOTTAB
SC.SMG to give the port a buffer on send failure
SC.RMG to get the number of buffers to queue
SC.ONL to get 2 buffers; 1 for .SBOBB and 1 for port
SC.ACB to get a buffer to use as the connect block
SC.BF2 to get buffers requested upon connect or accept

This routine is used to obtain message buffers from SCA's pool. First, the number of buffers desired is checked against FQCNT and if greater, the routine will return an SCSNBA error. Next, the number of buffers left after the request is satisfied (FQCNT-NUMBUF) is checked to see if it falls below the minimum threshold (MGTRSH). [MGTRSH is used because it is the constant lower threshold below which no requests for buffers will be granted unless they are small (less than LRGREQ). MINMSG is not used because it is a fluctuating value which depend upon the number of nodes up at the moment.] If there are enough buffers remaining (or if not and the request is small), the request is honored. If the request is honored because it is a small request, MBUST is incremented to record this. To honor the request, the message free queue (top is TOPFQ and bottom is BOTFQ) is traversed to insure that there actually are the required number of buffers.

If there is an inconsistency, a SCAMCR BUGCHK will result and FQCNT will be recalculated by tracing the free queue again. NMBCNT (the number of times we ran out of message buffers) and RMRCNT (the number of times allocation was refused) are incremented. DDCFSF is incremented to force SC.ALC to run. Also, ASRMR is updated to reflect the average size of refused requests. A SCSNBA error will be returned.

If the scan of the free queue is successful, TOPFQ, BOTFQ, and FQCNT are updated. If it is found that FQCNT has fallen below MINMSG, DDCFSF is set to allow SC.ALC to run.

If the request is refused, RMRCNT is incremented and ASRMR is updated to reflect the average size of refused requests. A SCSNBA error will be returned. This will occur when there are not enough buffers present to grant the request or when a large request forces the count below the buffer threshold.

o SC.ALD - Allocate datagram buffers from SCA pool

Called by: SC.SDG to give the port a buffer on send failure
SC.RDG to acquire the buffers to queue
SC.BF2 to get buffers requested upon connect or accept

This routine is used to obtain datagram buffers from SCA's pool. First, the number of buffers desired is checked against DFQCNT and if greater, the routine will return an SCSNBA error. Next, the number of buffers left after the request is satisfied (DFQCNT-NUMBUF) is checked to see if it falls below the minimum threshold (DGTRSH). [DGTRSH is used because it is the constant lower threshold below which no requests for buffers will be granted unless they are small (less than LRGREQ). MINDG is not used because it is a fluctuating value which depends upon the number of nodes up at the moment.] If there are enough buffers remaining (or if not and the request is small), the request is honored. If the request is honored because it is a small request, DBUST is incremented to record this. To honor the request, the datagram free queue (top is TOPDFQ and bottom is BOTDFQ) is traversed to insure that there actually are the required number of buffers.

If there is an inconsistency, DFQCNT will be recalculated by tracing the free queue again. NDBCNT (the number of times we ran out of datagram buffers) and RDRCNT (the number of times allocation was refused) are incremented. DDCFSF is incremented to force SC.ALC to run. Also, ASRDR is updated to reflect the average size of refused requests. A SCSNBA error will be returned.

If the scan of the free queue is successful, TOPDFQ, BOTDFQ, and DFQCNT are updated. If it is found that DFQCNT has fallen below MINDG, DDCFSF is set to allow SC.ALC to run.

If the request is refused, RDRCNT is incremented and ASRDR is updated to reflect the average size of refused requests. A SCSNBA error will be returned. This will occur when there are not enough buffers present to grant the request or when a large request forces the count below the buffer threshold.

o SC.CMG/SC.CDG - Create message/datagram buffers

Called by: SC.ALC to create buffers needed to boost above minimum
SC.BF2 to create buffers needed by connect or accept

Calls: SC.BBF to break memory pages into buffer chain

These routines are used to create message and datagram buffers. It is called when the SCA buffer pool is exhausted (or below minimum values) and needs to be restocked. The number of buffers returned is rounded up to fill an integral number of pages so extra buffers may be provided.

First the number of pages required to fill the buffer request is calculated. For each page required, EPGMAP is searched to locate an empty page. If no empty pages are found, the routine returns with a SCSNEB error. The search starts at offset LSTEPG (which is initialized to 777) and decrements towards the top of the map. When a free page is found, the next lower offset is placed in LSTEPG and a call is made to MLKMA to lock down the page. Finally, a call is made to SC.BBF to break the page into buffers of the appropriate size (C%MGSZ for messages and C%DGSZ for datagrams).

- SC.BBF - Break memory page into buffers

  Called by: SCA to create the initial chain of buffers
  SC.CMG to create chain of message buffers
  SC.CDG to create chain of datagram buffers

  This routine will take a page of memory and break it into a chain of message or datagram buffers. The page is broken up into buffers of C%BINV+C%MGSZ size for messages and C%BINV+C%DGSZ for datagrams. C%BINV is the size of the invisible area which is prepended to the top of each buffer for local use (it is never sent across the CI). The FLINKs of each buffer point to the next buffer in the chain.

- SC.RBF - Return a message buffer

  Called by: SC.ERR to return the two buffers obtained at SC.ONL
  SC.INT to return buffer sent by SCA
  SC.RCB to return buffers queued by connection
  SC.JSY to return buffers queued by JSYS
  SC.GCB to return buffers in RET_CREDIT field (.CBRTC)
  SCM.RM to return one of a chain of message buffers
  SC.ABF to return extra buffers from SC.CMG

  This routine will return message buffers to the SCA message free queue. First, the entire buffer (C%BINV+C%MGSZ) is zeroed. Then the buffer is linked to BOTFQ and FQCNT is incremented to reflect the additional buffer. Note that this routine is passed back to the caller by SC.ABF as the routine to call to return the buffer.

- SC.RLD - Return a datagram buffer

  Called by: SC.CRD to return buffer which was canceled
  SC.RCB to return buffers queued by connection
  SC.JSY to return buffers queued by JSYS
  SCM.RD to return one of a chain of datagram buffers
  SC.ALD to return extra buffers from SC.CDG

  This routine will return datagram buffers to the SCA datagram free queue. First, the entire buffer (C%BINV+C%DGSZ) is zeroed. Then the buffer is linked to BOTDFQ and DFQCNT is incremented to reflect the additional buffer. Note that this routine is passed back to the caller by SC.ALD as the routine to call to return the buffer.

```
;States of a connection. Names in all caps are from corporate spec.

        .CSCLO==1               ;Closed (CLOSED)
        .CSLIS==2               ;Listening (LISTENING)
        .CSCSE==3               ;Connect request was sent (CONNECT_SENT)
        .CSCRE==4               ;Connect request was received (CONNECT_REC)
        .CSCAK==5               ;Connect response was received (CONNECT_ACK)
        .CSACS==6               ;Accept request was sent (ACCEPT_SENT)
        .CSRJS==7               ;Reject request was sent (REJECT_SENT)
        .CSOPN==10              ;Connection is open (OPEN)
        .CSDSE==11              ;Disconnect request was sent (DISCONNECT_SENT)
        .CSDRE==12              ;Disconnect request received (DISCONNECT_REC)
        .CSDAK==13              ;Disconnect response received (DISCONNECT_ACK)
        .CSDMC==14              ;Waiting for disconnect response (DISCONNECT_MA

;States of a connection block

        .BSFRE==:1              ;Free
        .BSALL==:2              ;Allocate
        .BSCNP==:3              ;Connect pending
        .BSACP==:4              ;Accept pending
        .BSRPN==:5              ;Reject pending
        .BSCRP==:6              ;Credit pending
        .BSDPN==:7              ;Disconnect pending


;Messages

        .STORQ==0               ;Connect request
                .LNORQ==^D64
        .STORS==1               ;Connect response
                .LNORS==^D16
        .STARQ==2               ;Accept request
                .LNARQ==^D64
        .STARS==3               ;Accept response
                .LNARS==^D16
        .STRRQ==4               ;Reject request
                .LNRRQ==^D16
        .STRRS==5               ;Reject response
                .LNRRS==^D12
        .STDRQ==6               ;Disconnect request
                .LNDRQ==^D16
        .STDRS==7               ;Disconnect response
                .LNDRS==^D12
        .STCRQ==10              ;Credit request
                .LNCRQ==^D12
        .STCRS==11              ;Credit response
                .LNCRS==^D12

        .STAMG==12              ;Application message
        .STADG==13              ;Application datagram
```

;Table X - calls from sysap to SCA.
;Note: if new connection state is in <>, it isn't set until the pending
;message has been sent

Routine	State	Legal?	New block State	New connection State
SC.CON	Closed	Y	CONN_PEND	<connect_sent>
	Listening			
	Connect_sent			
	Connect_rec			
	Connect_ACK			
	Open			
	Accept_sent			
	Reject_sent			
	Disconnect_sent			
	Disconnect_rec			
	Disconnect_ACK			
	Disconnect_match			

Routine	State	Legal?	New block State	New connection State
SC.LIS	Closed	Y	-	Listen
	Listening			
	Connect_sent			
	Connect_rec			
	Connect_ACK			
	Open			
	Accept_sent			
	Reject_sent			
	Disconnect_sent			
	Disconnect_rec			
	Disconnect_ACK			
	Disconnect_match			

Routine	State	Legal?	New block State	New connection State
SC.ACC	Closed			
	Listening			
	Connect_sent			
	Connect_rec	Y	ACC_PEND	<Accept-sent>
	Connect_ACK			
	Open			
	Accept_sent			
	Reject_sent			
	Disconnect_sent			
	Disconnect_rec			
	Disconnect_ACK			
	Disconnect_match			

* * Old code didn't change state

Routine	State	Legal?	New block State	New connection State
SC.REJ	Closed			
	Listening			
	Connect_sent			
	Connect_rec	Y	REJ_PEND	<reject_sent>
	Connect_ACK			
	Open			
	Accept_sent			
	Reject_sent			
	Disconnect_sent			
	Disconnect_rec			
	Disconnect_ACK			
	Disconnect_match			

* * * Old code left state as "listen"

Routine	State	Legal?	New block State	New connection State
SC.DIS	Closed	Y	FREE	closed
	Listening	Y	FREE	closed
	Connect_sent	Y	FREE	closed
	Connect_rec	Y	REJ_PEND	reject_sent
	Connect_ACK	Y	FREE	closed
	Open	Y	DIS_PEND	disconnect_sent
	Accept_sent	Y	FREE	closed
	Reject_sent	Y	FREE	closed
	Disconnect_sent	Y	FREE	closed
	Disconnect_rec	Y	DIS_PEND	disconnect_match
	Disconnect_ACK	Y	FREE	closed
	Disconnect_match	Y	FREE	closed

;Table Y -- processing SCS work queue

Block State	Op code to send	New connection State
CONN_PEND	CONNECT_REQ	connect_sent
ACC_PEND	ACCEPT_REQ	accept_sent
REJ_PEND	REJECT_REQ	reject_sent
DIS_PEND	DISCON_REQ	-- (already done)
CREDIT_PEND	CREDIT_REQ	--

;Table Z - sending an SCS message

Name	Code	Length	priority	Response
CONNECT_REQ	.STORQ==0	.LNORQ==^D64	.MPCMM	CONNECT_RSP
CONNECT_RSP	.STORS==1	.LNORS==^D16	.MPCMM	
ACCEPT_REQ	.STARQ==2	.LNARQ==^D64	.MPCMM	ACCEPT_RSP
ACCEPT_RSP	.STARS==3	.LNARS==^D16	.MPCMM	
REJECT_REQ	.STRRQ==4	.LNRRQ==^D16	.MPCMM	REJECT_RSP
REJECT_RSP	.STRRS==5	.LNRRS==^D12	.MPCMM	
DISCONNECT_REQ	.STDRQ==6	.LNDRQ==^D16	.MPLOW	DISCONNECT_RSP
DISCONNECT_RSP	.STDRS==7	.LNDRS==^D12	.MPCMM	
CREDIT_REQ	.STCRQ==10	.LNCRQ==^D12	.MPFLO	CREDIT_RSP
CREDIT_RSP	.STCRS==11	.LNCRS==^D12	.MPCMM	

;Table K - receiving a message

Name	Code	Current State	Legal?	Response	New connection State
CONNECT_REQ	0	Closed	Y	CONNECT_RSP NM	
		Listening	Y	CONNECT_RSP	connect_received
		Connect_sent			
		Connect_rec			
		Connect_ACK			
		Open			
		Accept_sent			
		Reject_sent			
		Disconnect_sent			
		Disconnect_rec			
		Disconnect_ACK			
		Disconnect_match			

Name	Code	Current State	Legal?	Response	New connection State
CONNECT_RSP	1	Closed			
		Listening			
		Connect_sent			
		success	Y	-	Connect_ACK
		failure	Y	-	Closed
		Connect_rec			
		Connect_ACK			
		Open			
		Accept_sent			
		Reject_sent			
		Disconnect_sent			
		Disconnect_rec			
		Disconnect_ACK			
		Disconnect_match			

* * * Old code set state to "closed no match" if failure

Name	Code	Current State	Legal?	Response	New connection State
ACCEPT_REQ	2	Closed	Y	ACCEPT_RSP NM	closed
		Listening			
		Connect_sent			
		Connect_rec			
		Connect_ACK	Y	ACCEPT_RSP	Open
		Open			
		Accept_sent			
		Reject_sent			
		Disconnect_sent			
		Disconnect_rec			
		Disconnect_ACK			
		Disconnect_match			

* * * * Old code closed v.c. if state was closed; required block state of connect_pend, too

Name	Code	Current State	Legal?	Response	New connection State
ACCEPT_RSP		Closed			
		Listening			
		Connect_sent			
		Connect_rec			
		Connect_ACK			
		Open			
		Accept_sent			
		Match	Y	-	Open
		No match	Y	-	closed
		Reject_sent			
		Disconnect_sent			
		Disconnect_rec			
		Disconnect_ACK			
		Disconnect_match			

* * * * Old code expected state to be connect_received; set new state to closed; required block state of accept_pend

Name	Code	Current State	Legal?	Response	New connection State
REJECT_REQ		Closed			
		Listening			
		Connect_sent			
		Connect_rec			
		Connect_ACK	Y	REJECT_RSP	Closed
		Open			
		Accept_sent			
		Reject_sent			
		Disconnect_sent			
		Disconnect_rec			
		Disconnect_ACK			
		Disconnect_match			

* * * Old code set new state to closed-rejected
* * * Why not treat "closed" as legal as in accept_request

Name	Code	Current State	Legal?	Response	New connection State
REJECT_RSP		Closed			
		Listening			
		Connect_sent			
		Connect_rec			
		Connect_ACK			
		Open			
		Accept_sent			
		Reject_sent	Y	-	closed
		Disconnect_sent			
		Disconnect_rec			
		Disconnect_ACK			
		Disconnect_match			

* * * Old code expected state to be "listen"; left it as "listen"

Name	Code	Current	Legal?	Response	New connection
DISCONNECT_REQ		Closed			
		Listening			
		Connect_sent			
		Connect_rec			
		Connect_ACK			
		Open	Y	DISCONNECT_RSP	disconnect_rece
		Accept_sent			
		Reject_sent			
		Disconnect_sent	Y	DISCONNECT_RSP	disconnect_matc
		Disconnect_rec			
		Disconnect_ACK	Y	DISCONNECT_RSP	closed
		Disconnect_match			

Name	Code	Current State	Legal?	Response	New connection State
DISCONNECT_RSP		Closed			
		Listening			
		Connect_sent			
		Connect_rec			
		Connect_ACK			
		Open			
		Accept_sent			
		Reject_sent			
		Disconnect_sent	Y	-	Disconnect_ACK
		Disconnect_rec			
		Disconnect_ACK			
		Disconnect_match	Y	-	closed