Design specification for the Systems Communications Architecture

21-Sep-83

1.0 Introduction

1.1 Scope

This document is intended as a guide to SCAMPI and SCSJSY, the TOPS-20 implementation of the Systems Communications Architecture protocol. Any non-obvious code or algorithm used in either the monitor SYSAP support code or JSYS support code is described here. It is a non-goal to describe each routine in detail. The intent is to provide a map of major highways, not a street map.

The document is broken into two parts, one for SCAMPI and one for SCSJSY. Each could easily be a stand alone document but are included together here for consistency with other SCA documents and convenience.

1.2 Related documents

The reader should at least be familiar with the following documents before proceeding with this specification. The corporate SCA spec is particularly important as all terminology used in this document follows standards set in the corporate spec.

1. SCA Functional Specification, 19-September-83 (CDunn)
2. Systems Communications Architecture, 20-July-82 (Strecker)
3. LCG CI Port Architecture Specification, 11-July-83 (Dossa/Keenan)
4. TOPS-20 Coding Standard, 23-Mar-83 (Murphy)

2.0 SCAMPI (Monitor SYSAP support)

2.1 Major data structures

2.1.1 System block

The system block is the set of per system data for each remote known to the monitor. The set of system blocks for the local host can be accessed through the system block list (SBLIST). This is the sparse list of all system block addresses. Each entry is accessed by indexing into the list by node number. This will yield the address of the system block for that node number. In addition to SBLIST each system block has a forward link to the next block.

There is a special index into SBLIST. When -1 is given as a node number on a listen call, the SYSAP is indicating that any remote node is acceptable. Hence -1 is only meaningful when doing a listen call.

System block data includes the work queue, the connect block queue, and the work queue for this remote. The system block is mainly maintained by the port driver but there are cells maintained by SCA. These cells are described in [1].

2.1.2 Connect Block

The connect block is the per connection database and is entirely maintained by SCA. All connect blocks are linked onto two doubly linked queues. One of these lists is the set of connections for a particular system block. The other list is the set of connections owned by a particular fork. This list and its uses are described in the second half of this document in the section on SCSJSY. Further description of the cells in this block may be found in [1].

2.1.3 Remote system work queue

These queues (one per known remote) are used for the stacking of SCA overhead messages to remotes. Since the protocol allows only one outstanding overhead message at any time for each remote, the overhead messages which SCA desires to send while there is one already outstanding are placed on this queue. When the response to the outstanding message comes in, the next message on this queue is sent. Note that the head and tail pointers for these queues are stored in the system block for each node.

2.2 Overview

SCAMPI can be broken down into a number of major areas, routines that process requests from SYSAPs, interrupt handlers, periodic function handlers, buffer management routines, and general support routines.

2.2.1 SYSAP request handlers

The request handlers are the most simple and straight forward part of SCAMPI. These routines are most easily understood by reading the code, rather than a verbal description here. There are however some rules which are followed in this set of routines.

The more important rule is that interrupt level may come along and change connection data out from under the nose of a request handler. Hence an interlock scheme must be arranged for places that cannot tolerate such a change. For all data except the connect state, the CION/CIOFF macros may be used to interlock connection data for periods of a few instructions. Since the connection state needs to be interlocked for very long periods (perhaps up to a millisecond) another interlock scheme is used. The SCON/SCOFF macros are used to interlock the connect state. These macros call the routines SC.SOF and SC.SON in SCAMPI which will do nesting and interlock checking much like that done by CION/CIOFF. See the sections on SC.SON and SC.SOF for more details on these routines.

There is one other interlock scheme used for single instructions that wish to increment fields in the connect block. In particular when SCA wishes to update the credit fields in the connect block, an ADDM AC,CBLOC is done where AC contains the number of additional credits (or negative for returning credit) and CBLOC is the address of the full word credit field in the connect block. Since we can be guaranteed that the instruction will complete once started we can be sure that the instruction need not be interlocked with CION/CIOFF.

The second rule is that the SYSAP is trustworthy. Hence minimal checks are done for SYSAP argument validity. Only data items over which the SYSAP has no direct access (E.G. connection state) are checked.

2.2.2 Interrupt handlers

There are a number of entry points in SCA for the port driver to call at interrupt level. Each of these entry points makes a few assumptions about what has happened and what it can do.

A common assumption made by all interrupt handling routines is that they may change certain data items in the connect block. In particular the the receive credit level may be changed at interrupt level. The connection state on the other hand is a special case. The problem is that we may very well be interrupting a piece of code that checked the state of the connection, saw that it was open, and is about to send messages or datagrams based on this check. The connection must not be closed until after this packet send is complete. Hence there is an interlock mechanism for the connection state. The SCOFF/SCON macros turn off/on the ability to change the connect state. When the SCON is done by the sensitive routine, a support routine is called to check a connection specific queue for change of state request blocks. A part of this block is the address of a chain of packets that needs to be sent along with the change of state.

A specific case is the sending of a disconnect request. When a remote sends the local host a disconnect request, the interrupt handler for the message type checks the interlock for the connect block state. If it is unlocked it proceeds to send the rest of the handshake. If it is locked however, it puts an entry onto the connection's block state queue along

with the address of the chain of messages to send, (in this case the disconnect response followed by the disconnect request) and returns. When the routine which locked the connection state releases the lock, the queue is checked for change of state entries. From the remotes point of view, an application packet will be received for a connection which has sent the first part of the disconnect handshake. Hence it must drop this packet without closing the VC.

2.2.2.1 SC.ONL

At this entry point SCA assumes that a new VC has just been established. It may be a reopening of an old VC or the first circuit to a new node. In either case we have the node number of the system newly arrived and we may go ahead and perform SYSAP notifications. These notifications are different from all others in that it happens on a per SYSAP basis rather than a per connection basis. There was no reason to notify the same SYSAPs once for each connection it owns about a new node. Hence the SYSAP notifies SCA of an address to call when this event occurs. Typically this address is the same address given for all other notifications, but that is a decision for the SYSAP designer. Each SYSAP notified requires the event code for new circuit and the node number of the newly aquired node.

2.2.2.2 SC.ERR

This is the entry point for circuit closure. If a circuit is closed for any reason, SCA expects the port driver to call this entry point. This includes the case of SCA requesting a circuit closure. It should be noted here that SCA must request the opening of a circuit after it has been closed. This is accomplished through the use of the SCOFF/SCON macros. Upon receiving notification of circuit closure, SCA must close all open connections on that circuit. It cannot change the state of a connection which has the connect state interlocked however. A circuit wide count of interlocked circuit states is maintained. When a SCON macro is performed, the count has reached zero and the flag for "VC requires open call" is set in the system block, then we may call the port driver to open the circuit again. While SCA is here, we must notify all SYSAPs that the circuit has gone away. This implies that the SYSAP must be able to deal with a circuit going away during an SCA call.

2.2.2.3 SC.OFL

This entry point is identical to SC.ERR except the code used on SYSAP notification is for node offline rather than circuit closed on error.

2.2.2.4 SC.DMA

This entry point is called when a named buffer transfer has completed. All that needs to be done here is notify the connection that requested the transfer.

2.2.2.5 SC.INT

This is the major interrupt level entry point for SCA. It is from this point that SCA handles all incoming application and SCA packets. A dispatch to the correct routine is done based on the SCA message type. Other than the general interrupt level cautions outlined above, nothing very special is done here.

2.3 Periodic function handlers

Within SCA there are a few functions that are handled on a per time basis. At present these functions are idle chatter, the management of buffer allocation levels, and the deletion of connection data. All SCA time driven activity happens during the one hundred millisecond clock cycle of the scheduler. Hence there is a clock counter and instruction to execute in STG for the SCA clock. The instruction is a PUSHJ into SC.CLK. This routine then does simple checks to see if it is time for any SCA events to happen, and dispatches to the correct handling routines when the events are due.

2.3.1 Idle chatter

Idle chatter is defined as the polling of remote nodes with connections open to them. This is done since most port hardware implementations can answer a request ID without software intervention. Hence the port driver poller will not declare a node offline until it stops answering request IDs on both paths. Since the port can stay up for extremely long periods afer the software has crashed, detection of remote software failure could potentially take a very long time. Hence SCA will send a credit request for zero credit to any remote which has a connection open to it. These credit requests are only sent if the node has not sent us a packet for over five seconds. If a credit request is sent, the response turn around is timed. If the response arrives within five seconds, the node is online and SCA does nothing except turn off the timed message flag in the system block. If the response has not arrived after five seconds, the node is declared down and circuit closure is requested.

2.3.2 Buffer allocation level management

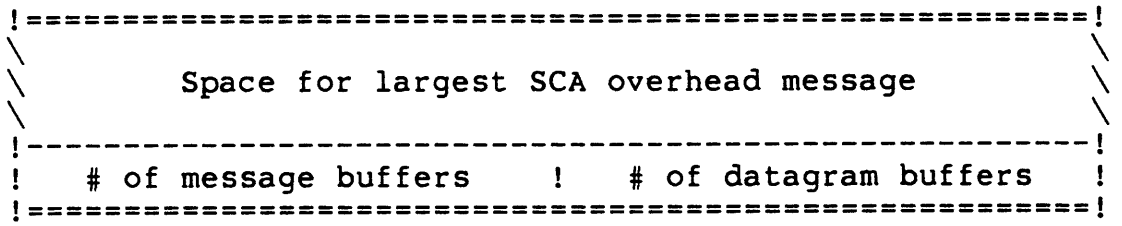
Buffer levels are handled with a twofold process. First, a threshold level is maintained, below which SYSAPs may request buffers without waiting. If a SYSAP directly calls a buffer allocation routine (SC.ABF/SC.ALD) with a request for more than what is on hand, the failure return is taken and no further action is taken by SCA. In general it is assumed that for cases other than connection startup and packet reception buffers SYSAPs will only make small buffer requests (between 1 and 3 buffers). These requests will generally present no problem, as the threshold level should be well above this.

Second, if a connect request, listen request, accept request, or receive buffer request is made with a large numbers of buffers specified, SCA will try to fill the request immediately. If this request would put us under threshold, then the request is defered until job 0 can allocate the buffers. All of this is transparent to the SYSAP however. The returned information (if any) is the same whether the buffers were available or not. If the buffers were not available, then whatever the SYSAP requested will take longer to finish. Since no SYSAP should time these requests, this does not present a problem.

Internally, SCA notices that the request for buffers will put us under

threshold and hence this request must be defered. Hence the connect request message is placed on a queue, hung off the system block. Note that at this point the connect request has been completely assembled and is ready for transmission. In addition to the normal connect request data, a short block of data used for creating the required buffers has been appended to the connect request. After the entry has been made to the queue, job zero is poked to run and check for this type of request. Once job zero has created the required buffers, it will either call SC.SCA to send a request, or will simply return the buffer to the SCA free pool if this is a buffer queue request.

The following is the format of the data block used to describe the allocation of additional buffers:



.BZNUM
This word defines the number of buffers to be allocated.

Note that if there are message buffers being created, the message being sent here is a credit request. Hence the count of message buffers plus the current pending receive credit must be placed in the credit field of the packet. If there are datagram buffers being added here then the count of datagram buffers in the connect block must be updated to include these new buffers.

2.3.3 Connection data deletion (SC.RAP)

This periodic function is responsible for the return of resources associated with aborted or disconnected connections. Once the abort bit has been set by some other part of SCA, the next pass of this function will light the reap bit. On the following cycle, SC.RAP will check the count of packets outstanding on the KLIPA command queue and delete the connect block if the count is zero. It should be noted that this count of packets on the queue only accounts for packets that the software gets back. I.E. only those packets which do not end up on a free queue are counted. Packets which end up on a free queue are not seen by the software after the send has completed and hence cannot be counted. There is a substantial delay however (minimum of 15 seconds) between the abort of the connection and the deletion of the data. This should allow enough time for those packets not counted to make it off the KLIPA command queue.

2.4 Buffer management routines

These routines handle the allocation and return of SCA free pool buffers. The SCA free pool is a singly linked list of the buffers available. There are two such pools, one for datagrams and one for messages. The return of buffers to these pools is a simple operation where the buffer is placed at the end of the list and the buffer pool count incremented as necessary. The allocation of these buffer is a bit more complex however.

When SCA is called upon to allocate a number of buffers (N) from one of the free pools, there are a number of checks that need to be done. First, if there are not enough buffers to fill the request (as determined from the buffer count) the request is rejected outright. If there are enough buffers but the number of buffers remaining in the pool is less than a threshold value, then the request is granted and flags are lit for job 0 to add more buffers to the free pool. The number of buffers added is based on the number of nodes seen the in network plus a few additional buffers. If during the process of allocating the buffers, it is determined that the buffer count is wrong and there are in fact not enough buffers to fill the request, the partial allocation is returned to the free pool and the request is rejected. After this process the free pool count is correct since the allocation kept track of how many buffers were in the partial allocation.

2.5 General support routines

These routine perform the common functions required to support SCA. In general these are very simple routines that need no explanation. Only those which make assumptions that are not obvious, or those that use non-obvious algorithms shall be called out.

2.5.1 Connection data deletion

When an open connection has been disconnected or any connection has been aborted, the database for that connect must be deleted. This is done by a number of routines whos' function may not be completely obvious. In general the idea is to wait for all outbound traffic to complete before deleting connection data. This can be done since the handling of inbound packets is identical whether the data is around or not. Outgoing traffic is easily handled since there is a count of outstanding messages on the port queues in the connect block. When this count goes to zero all outgoing traffic has completed. We are guaranteed that there will be no further traffic since SCA will not try to send packets on a connection in any state but open. Once this count has gone to zero, the data may be deleted.

To insure that the KLIPA has had time to complete transmission of any packets outstanding for this connect, SCA waits a while before checking the counts. The periodic function to check reap and abort bits will then clean up the connect data and all resources associated with the connect. For more detail on this process, see the section on periodic functions.

2.5.2 SC.SON/SC.SOF

These routines interlock the connect state of each connection. There is an interlock word in the connection block that when positive is an indication that the lock is locked and someone wishes to lock it. When zero the lock is locked but there are no other requests to lock it. When -1 the lock is available. When interrupt level discovers that it would like to change the state of the connection but the lock is not available, an entry is made to a system wide queue, indicating the new state and a chain of packets that need to be sent as a result of this new state. When the unlock is done this queue is checked for entries for this connection. If entries are found they are processed immediately.

2.5.3 SC.PON/SC.POF

These two routines are responsible for the turning off and on of CI interrupts. This is accomplished through the turning off/on of the PI channel for the KLIPA. The following algorithm is used to determine whether the channel state should be changed.

- SC.POF -

Fig. 1

- SC.PON -

Fig. 2

3.0 SCSJSY (User mode SYSAP support)

3.1 Overview

SCSJSY provides a user interface to the CI through SCA. To do this it does nothing more than translate user requests into SCA SYSAP request handler calls and turns SCA callbacks into a PSI for a fork. A major part of this job is the careful checking of user arguments. Since SCA assumes that SYSAPs are trustworthy, a bad argument to an SCA call can cause a system crash or CI failure fatal to a circuit. Hence SCSJSY must take care to insure the integrity of user arguments.

In general, SCSJSY handles two types of traffic, inbound, and outbound. Inbound traffic is placed on fork and connection lists and the owning fork gets a PSI if the list was empty before this packet arrived. Outbound traffic is placed on KLIPA command queues through the use of SCA calls and blocks until the confirmation of send complete comes up from SCA. This blocking is necessitated by the way packet transmission is done. Details of this process can be found in the section on JSYS level packet transmission.

The structure of SCSJSY can be broken up into two parts, a part that handles service requests from the user, and a part to handle user notification of interrupt events.

3.2 Major data structures

3.2.1 Connect blocks

In addition to the data for monitor SYSAP support, the connect block contains a number of cells for JSYS SYSAP support. These include user buffer counts and list head/tail pointers for various per connection lists. In general these lists are all doubly linked, and each piece of data is on two of the lists. In particular each piece of data is on a fork list and on a connection list. This enables JSYS SYSAPs to request data on a fork wide basis or a per connect basis. This is greatly simplifies management of multiple connections. It also means that when data items are added or removed from its lists, they will very likely be pulled from the front of one list and the middle of the other. All support code must be prepared to deal with this case.

A side effect of having data available on a fork wide basis is that the head/tail pointers for the lists appear in the PSB. As a result, various places in the code will want to modify the PSB of another fork, or in the case of interrupt level, modify non-resident memory. To solve this problem a system wide list was created to store data items until process level could be summoned to modify non-resident memory. The details of this process can be found in the section on interrupt level processing.

Connect blocks do not present this kind of problem since they are always resident. They do however appear on a chain of connect blocks available to the fork. Hence when the first or last connect block on the list is deleted, the PSB of the owning fork will be modified. Hence this process also happens in process context. The details of the connection block deletion process can be found in the section on monitor SYSAP support.

3.2.2 SCA data section

To ease the allocation of memory, SCA has a virtual section all to itself. This section has two uses, a place to allocate buffer space, and a place to map user packets for transmission. Buffer space is allocated starting at the end of the section and grows down into memory a page at a time. User packets being made ready for transmission are mapped into the begining of the section. This scheme allows the section to be used for both purposes and still have a dense set of pages for each.

3.3 Service request handlers

Unless otherwise noted, service request handlers check arguments, translate required arguments into format for passing to SCA, call SCA, and return. If any data needs to be passed back to the user program that is not returned by the call to SCA, an event is generated and placed on the fork/connection event queue.

3.3.1 Send a packet (SCSSMG/SCSSDG)

A major consideration in sending packet data was the performance loss caused by copying data from user space to monitor space. Hence SCSSMG/SCSSDG maps the packet into the low end of the SCA section. This is the why the set of restrictions (detailed in [1]) is placed on packet sends. Since the SCA header must be contiguous with the packet text, there must be room for the header between the page boundry and the start of the SYSAP text. This is why the SYSAP specifies the base address of the SCA header and not the base of the SYSAP text when talking about buffers. It insures the SYSAP understands that room for the header must be available.

3.3.2 Read port counters

To read the port counters the port driver must place a command on a KLIPA command queue and notify SCA when the response comes back on the KLIPA response queue. Hence the SCA call will not return directly with the requested information. In this case however the JSYS call will block until the information has been returned. This alleviates the need for yet another event type, and since this is not really an event but a request for information, it returns the data immediately just as the other information calls do. It just has to wait a while to return the information. Care must be taken however to not be NOINT wile doing this. This will allow the user to get out of a hung call to SCS%. This implies that the routine which returns this information must be able to handle the connection going away when the data actually comes back. It also means no resources can be owned when the dismiss is done.

3.4 Interrupt event handlers

The operation of the interrupt handlers is actually rather simple. Each mearly records a small amount of data into a block and places the block onto a system wide list. It is the handling of this list which is rather circuitous:

1. Interrupt level places entry on system wide queue and zeros location SCSTIM. This location is the clock counter for a 100ms scheduler clock. SCHED will notice that the count is zero and XCT a CALL SCSCLK on the next 100ms cycle.

2. Upon arrival at SCSCLK the entire system wide list is scanned. The target fork number is retrieved from each entry and a bit is lit in FKSTAT for this fork.

3. The main scheduler loop notices that FKPS1 is on in FKSTAT and will end up at PIRQ. PIRQ checks the SCS% bit in FKSTAT and will call SCSPSI if the bit is on. Once we arrived at PIRQ we are in process context for the target fork. Hence we have the PSB we will modify locked in core.

4. SCSPSI will now scan the system wide queue for entries bound for the current fork. The entries it finds are placed on the fork and connection queues of the target fork. If the list in question was empty before the new entries were added, then a PSI is generated on the appropriate channel for the target fork.

When a user receives a PSI, SCS% functions to retrieve data are done to return information about events or to retrieve data from packets received.