MCCS

The Design and Implementation of a Multi-Computer Communications System

by

SHELDON LEE FOX

B.S., Kansas State University, 1973

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

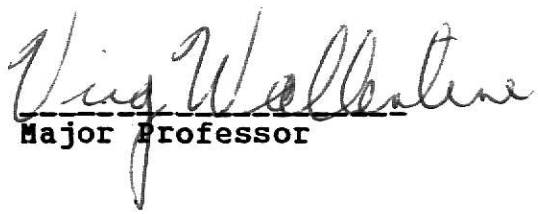
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1976

Approved by:



Major Professor

TABLE OF CONTENTS

FIGURES AND TABLES

ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. Virgil Wallentine, for his help and guidence throughout this project and Drs. Kenneth Conrow and Fred Maryanski for their comments and suggestions concerning this paper. I would also like to thank the Kansas State University Computing Center for letting me use their binary synchronous lines and modems for testing the bisync support in MCCS along with the technical consultation provided by the Systems Programming staff there throughtout the implementation period. Finally, I would like to thank the University of Waterloo, and more specifically, Bruce Uttley, for making available their version of the CMS SCRIPT text formatting program which made the preparation of this paper many times less difficult than it would of been using other methods.

I. INTRODUCTION

This paper describes the Multi-Computer Communications System (MCCS). MCCS allows tasks executing in one computer to communicate, via "messages", with tasks executing in one of several other computers.

MCCS provides two process-level primitives, SEND and RECV, to facilitate inter-process communication. MCCS is simple enough to allow it to be implemented easily on a mini-computer which provides a minimal level of hardware and software support using the asynchronous line protocol. On the other hand, employing the MCCS binary synchronous (BSC) line protocol [IBM01] on a more sophisticated computer system allows the speed and error detection/recovery facilities of BSC to be utilized to achieve more efficient communication.

MCCS message protocol was designed to be straightforward and does not depend on the existence of any complex operating system functions. Although this somewhat restricts the flexibility of the message system (or at least shifts some of the burden to the user process), it makes MCCS simple enough to be implemented on nearly any computer which supports the execution of one or more processes and can have a teleprocessing (TP) line attached to it. The original design goal of providing the capability for inter-process communication on multiple computers may then be realized.

Normally executing as a priviliged system task, MCCS should run without operator intervention. The procedures for automatic connection and disconnection of computers to one another have been defined. MCCS also handles any error conditions that should arise during its execution.

A message network employing MCCS consists of two or more computers, each of which may have connections, and therefore communicate, with one or more of the others. MCCS places no restrictions on the size and complexity of the network; it is instead restricted only by the hardware and software limitations of the computers which compose the network.

This project (and hence the paper) was motivated by a desire to provide a means of communications between an existing mini-computer (a NOVA 2/10) and a maxi-computer (an IBM 370/158). One of the major constraints placed on this communication was that the method of communications, whatever it be, require as few hardware and operating system modifications as possible. It was decided to utilize existing teleprocessing lines designed to support

teletype-like terminals. This was done for two reasons. Firstly, the hardware and operating system TP drivers already existed on the 370 and the TP drivers on the NOVA so they could be copied or used directly in MCCS. Secondly, it meet the original constraint of requiring minimal changes to the existing configuration.

Once the original desgin work had been done, it became obvious that there was a need to generalize the system to take on a much broader scope than just communications between a NOVA and a 370. Several applications requiring a generalized inter-computer communications facility appeared so it was decided to make the design changes necessary to support multiple computers in the configuration. These changes also included consideration for the implementation problems expected on different types of computers other than the two that were immediately involved.

Some of the applications for the message system (in the area of data base management) required large amounts of data be transmitted. Therefore it was decided to add support for BSC TP lines since much higher transmission speed and error detection and recovery could be then be obtained. (Even with the addition of the BSC support, the original constraints could still be met. BSC hardware support existed on the 370 and sources of software drivers could be obtained from Data General for the Nova.)

The following list shows some commonly used system functions which can be achieved by providing the appropriate set of processes[1] communicating using MCCS.

1) Machine synchronization using operating system tasks to coordinate the machines by the sending and receiving of messages.

2) Task synchronization (on either an inter- or intra-computer basis) by proper application of SEND and RECV primitives.

3) Buffer management.

4) The management of teleprocessing resources.

Part II contains the general design specifications for MCCS. It should serve both as a users' guide for someone wishing to use MCCS and as an implementation guide for someone

---

[1] This paper does not attempt to define these processes any further. They are mentioned here only to suggest applications for which MCCS might be used.

wishing to develop a version of MCCS on another computer.

Part III deals with an actual implementation of MCCS on an IBM 370/158 running VM/370 with CMS.[2] (It is referred to as MCCS/370 when it is necessary to distinguish it from other implementations of MCCS.) Someone familiar with IBM's 370s, teleprocessing, and VM/370 should be able to utilize Part III, along with the appendices, to operate, maintain, or enhance MCCS/370.

---

[2] Although not described in this paper, a version of MCCS has been implemented on a 32K (word) Data General NOVA (running RDOS) by Lee Allen, Richard McBride, and Jim Ratliff of the Kansas State University Computer Science Department. It was used in the initial testing of the asynchronous line protocol in the version of MCCS implemented on the 370.

II. SYSTEM SPECIFICATIONS

Part II describes the general specifications for MCCS and is intended for use by someone who wishes to implement a version of MCCS and needs to know the details necessary to insure the implementation is compatible with existing MCCS systems.

Sample control block format and execution flow descriptions will be given as well as the format that must be used for message transmission. The protocol used with both asynchronous and binary synchronous communication, including error detection and retry procedures, is also shown.

Process-Level Interface.

An executing process wishing to utilize MCCS does so by invoking the MCCS SEND or RECV routines. A high-level language CALL statement similar to that shown in Figure 1(a) will serve to illustrate the parameters necessary to send a message.

TO_ID specifies a six character process-identifier[3] of the process to which the message to being sent. The first two characters of the TO_ID designate a particular computer with which MCCS has a communication link (therefore each computer must be assigned a unique two character id). The last four characters (at least one of which must be a non-blank character) designate a particular process on that computer. If the first two characters specify a computer that MCCS does not have a link to, the request to send the message will be ignored and the TO_ID is set to all blanks before returning in order to notify the process of the invalid specification.

---

[3] All IDs are transmitted as ASCII characters. The SEND and RECV routines, however, may perform translation to and from ASCII if the processes sending and receiving messages normally use another character set, such as EBCDIC.

```
CALL SEND (TO_ID,MESSAGE,MSG_ID,USER_ID)
```

(a)

```
CALL RECV (FROM_ID,MESSAGE,MSG_ID,USER_ID,STATUS)
```

(b)

Figure 1: SEND and RECV Calling Format

MESSAGE specifies a 128 byte(*) area containing the information to be sent. The content and format of this message are not examined by MCCS so any necessary translation or reformatting must be done by mutual agreement of the communicating processes. The message is simply treated as a stream of binary data and transmitted to the receiving process unaltered.

MSG_ID specifies a 16 bit field that will be set to an integer upon return. This integer will uniquely identify the message that is sent. It is also available to the receving process for examination when the message is received. MSG_ID, when used in conjunction with USER_ID, provides the two processes a means of coordinating messages and responses if they so desire.

USER_ID specifies a 16 bit field that will be transmitted along with the message and made available to the receiving process. (If USER_ID is all binary zeros, the value of MSG_ID will be used for the value of USER_ID, otherwise, it is sent exactly as specified.) This field, with prior arrangement between the communicating processes, may be used to specify the content and format of the message, sequencing of messages, message priorities, return codes, etc.

The parameters needed to invoke the MCCS RECV routine are shown in Figure 1(b) and described below.

FROM_ID specifies a six character process-identifier. If it is all blanks, the first message in the list

---

* All references to a byte or character within MCCS designate a field containing eight bits.

cf messages that have been received by MCCS for this process is retrieved. If FROM_ID is non-blank, the first message in the list which was sent by a process with an identifier equal to FROM_ID will be retrieved. (See the explanation of USER_ID below for additional message selection criteria.) Upon return to the process from RECV, FROM_ID will be set to the process-id of the process which sent the message, or to blanks if the message could not be found or if there is no MCCS link to the computer specified by the first two characters of FROM_ID. (See the description cf the STATUS parameter below.)

MESSAGE specifies a 128 byte field into which the received message will be placed if found.

MSG_ID specifies a 16 bit integer into which is returned the MSG_ID of the message received.

USER_ID specifies a 16 bit field which functions in a manner similar to FROM_ID in that it may be used to select specific messages for retrieval. If it is all binary zeros, the first message on the list cf messages received for the process will be retrieved (subject to the restrictions imposed by FROM_ID). If it is non-zero, only a message with a matching USER_ID (set by the USER_ID parameter cf the call to SEND which created the message) will be retrieved (again subject to the restrictions imposed by FROM_ID). Upon return from the RECV call, USER_ID is set to the USER_ID cf the message that is retrieved. By specifing a non-blank FROM_ID and a non-zero USER_ID on the call to RECV, a process may selectively receive messages instead of receiving them in the order that they were sent.

STATUS designates a 16 bit field that is examined only when FROM_ID specifies a computer to which MCCS has a link but there is no message from that computer sent to this process with the specified FROM_ID and USER_ID. In this case, if STATUS is non-zero FROM_ID is set to blanks and control immediately returns to the process. If STATUS is zero the process is suspended until such time that a message is sent to it and is subject to retrieval based on the values of FROM_ID and USER_ID.

System Overview.

Figure 2 gives a conceptual view of the routines in MCCS (the left and right columns) and the data structures which they access (the middle column). This diagram is oversimplified (it doesn't show the interrupt structure necessary for BSC timeouts, for example), yet it is sufficient for a general discussion of control and data flows with MCCS. The dotted lines represent execution flow between the routines and the dashed lines represent data flow between the routines and the data structures with the arrowheads showing the direction of flow.

This diagram obviously does not represent the only way in which MCCS may be implemented. It should, however, provide a general guideline that can be modified to fit within the constraints imposed by most installation environments.

Data Structures.

A brief discussion of the contents of the MCCS data structures is given here -- a more detailed description is given in the discussion of the routines which modify them.

WAIT_LIST is a linked list describing all the processes which have issued a SEND or RECV request which has not been processed. (The reasons for not having processed the requests is given in the description of the Process-Dispatcher that follows.)

RECV_LIST is a linked list of all messages from another MCCS which have not yet been "received" (the process to which they were sent has not yet called RECV to accept the message).

SEND_LIST is a linked list of messages which have been created by processes issuing a call to SEND and have not yet been transmitted to the proper MCCS via one of the TP lines.

The Line Control Blocks describe the TP lines which provide the link between the MCCS on one computer and the MCCSs on other computers for which there is a communications path available. One line-control block exists for each TP line. They contain: 1) the two character identifier (ID) used to identify the computer to which this line is connected, 2) a buffer used to hold the message currently active on the line, 3) line

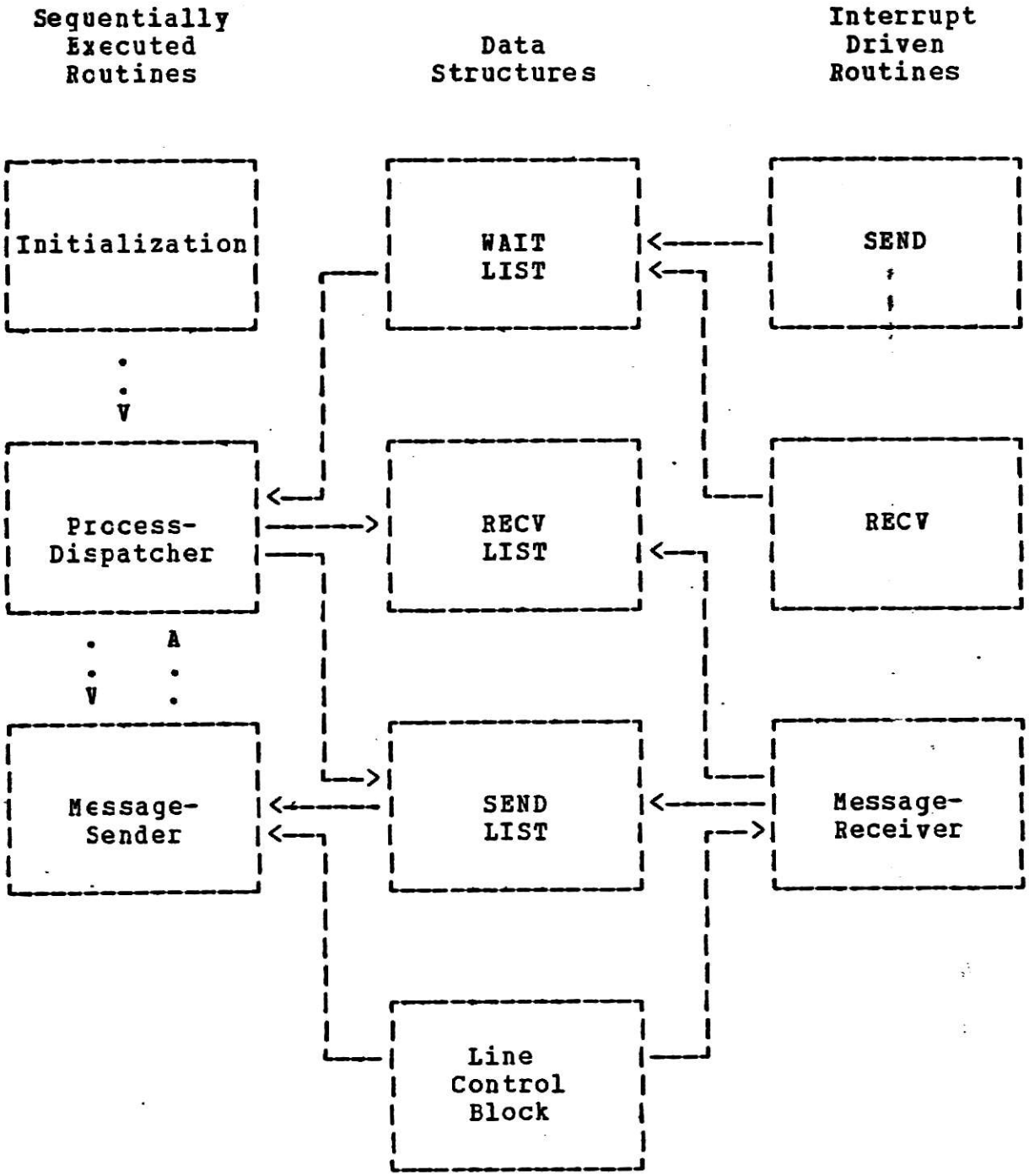


Figure 2: MCCS Control and Data Flow

status information, and 4) any device dependent information needed to maintain the TP line.

Functional_Description.

The Initialization routine is the first MCCS routine to gain control. It performs any control block formatting or initialization that is necessary. (It will probably access all MCCS data structures although this is not shown in Figure 2.) The Initialization routine also performs any communication with the operating system that is necessary to gain control of the TP lines, set up interrupt handlers, etc., as well as enabling the lines for communication.

Another important function performed by the Initialization routine is check-point recovery. It is highly desirable for MCCS not to lose any messages should a system failure occur, whether the failure is in MCCS, the operating system, or the hardware. This is necessary because a sending process shouldn't have to be suspended until it has been verified that the receiving process has accepted the message. This activity could very well take a considerable amount of time, time that the sending process could use to perform other functions. Assuming the sending process now does not know when its message has been successfully delivered, it therefore cannot be expected to maintain a copy of the message it last sent. The burden of maintaining the copy of the message is therefore shifted to MCCS. If MCCS goes down, it must be able to recover all control blocks on WAIT_LIST, SEND_LIST, and RECV_LIST when re-starting. Since these lists will normally be kept in memory for efficiency, a backup copy that represents the current contents of each list is maintained on secondary storage. It may then be recovered after a system failure and subsequent restart. A check-point should be taken each time a list is changed and the previous check-point destroyed so that the current copy is always available for re-starting if needed.

The Initialization routine determines if any check-point data exist, and if so, use it to set the initial contents of WAIT_LIST, SEND_LIST, and RECV_LIST to what they were immediately before the system failure.

After all initialization has been performed, the Initialization routine enters a loop which repeatedly calls the Process-Dispatcher and the Message-Sender. Although this could be a continous loop -- a "busy wait" loop -- it is more desirable to have the loop terminate after each iteration, proceeding only when there is more work for the Process-Dispatcher and Message-Sender to perform. The

interrupt driven routines (SEND, RECV, and the Message-Receiver) can then inform the Initialization routine that they have modified one of the data structures and that there is more work to be performed and thus the loop needs to be activated.

The Process-Dispatcher examines WAIT_LIST for Process Request Blocks (PRQBLOKs, see Figure 3). PRQBLOKs are allocated and added to the end of WAIT_LIST by SEND or RECV when called by a process. PRQBLOKs contain all the parameters specifed on the SEND or RECV call and sufficient information about the state of the process so that it can be suspended and restarted at some later time.[5]

If the Process-Dispatcher encounters a PRQBLCK for a process wishing to send a message, the TO_ID is examined. The first two characters are matched against the IDs in all the Line Contrcl Blocks to make sure there is a link to the computer the ID specifies. If a match is not found, the process's FROM_ID parameter is set to blanks to indicate it was invalid and the process is restarted.

When the TO_ID is valid an attempt is made to allocate a buffer to hold the message and the associated information (called a MSGBLOK, see Figure 4). If there is no buffer space, the PRQBLOK is skipped and the next one examined, effectively leaving the process suspended until buffer space is made available. Assuming the buffer is allocated, the MSGBICK is constructed using the TO_ID, MESSAGE, and USER_ID parameters from the call to SEND. The FROM_ID for the message is set to the ID of the process. (This is obtained by concatenating the two character ID of the computer on which MCCS is running with a one to four character identifier which uniquely defines the process which sent the message.) The message is assigned a unique number and it is placed in the MSGBLOK and the process's MSG_ID parameter. The MSGBLCK is then added to the end of SEND_LIST and the

---

[5] A process is suspended after SEND or RECV builds the PRQBLOK from the process's parameters. The process can be suspended in one of several ways. One way might be, assuming the operating system MCCS is running under supports the equivalent of the "P" and "V" primatives cr OS/360 POST and WAIT macros, is to make SEND and RECV reentrant. They may then, after building the PRQBLOK, add to it the semaphore/ECB address and then perform a P or WAIT. The Process-Dispatcher need only perform a V or POST using the semaphore/ECB address from the PRQBLOK to cause the SEND or RECV routine to resume, set the return parameters, and return control to the process.

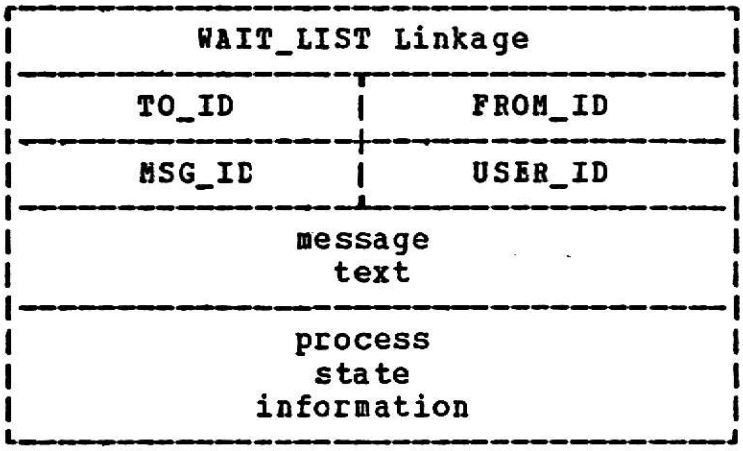


Figure 3: PRQBLOK format
(not to scale)

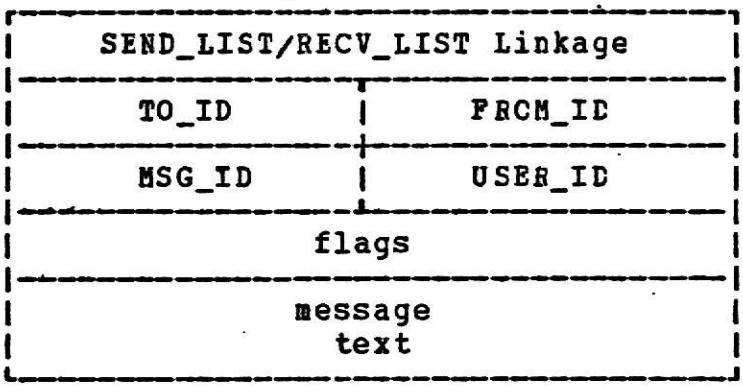


Figure 4: MSGBLOK format
(not to scale)

process allowed to resume.

Then the PRQBLOK is removed from WAIT_LIST and freed and the next PRBLOK examined.

A very useful extension to MCCS is easily realized at this point.[6] By allowing the TO_ID to begin with the same two characters as the ID of the computer from which the message

---

[6] This extension has been implemented in the version of MCCS written to run on the IBM 370 (described in Part III).

is sent, a process may send a message to a process on the same computer. This is implemented by checking for this condition and placing the MSGBLOK on the end of RECV_LIST instead of SEND_LIST.

If a PRQBLOK indicates the process wishes to receive a message, the MSGBLOKs on RECV_LIST are examined. (They are placed there by the Message-Receiver or by the extension noted above.) The process's ID is concatenated to the ID of the computer on which it is running and compared with the TO_ID in the RECV_LIST MSGBLOKs. If a match is found, the process's FROM_ID and USER_ID parameters are compared with those in the MSGBLOK. As described earlier, if the process's FROM_ID is non-blank, it must match the FROM_ID in the MSGBLOK. The process's USER_ID, if non-zero, must also match the USER_ID in MSGBLOK in order for the message to returned to the process. If it is to be returned, the process's FROM_ID, MESSAGE, MSG_ID, and USER_ID are set from the MSGBLOK. The process is then restarted, the MSGBLOK removed from RECV_LIST and the buffer released, the PRQBLOK removed from WAIT_LIST and freed, and the next PRQBLOK examined.

If all MSGBLOKs are examined and none is eligible to be retrieved, the process's STATUS parameter is checked. If it is zero, the process wishes to be suspended until it receives the message. This is done by simply leaving the PRQBLCK on WAIT_LIST and examining the next PRQBLOK. If the process doesn't wish to wait (STATUS is non-zero), its FROM_ID parameter is set to blanks to indicate the message was not found, the process restarted, the PRQBLOK removed from WAIT_LIST and freed, and the next PRQBLOK examined.

After the Process-Dispatcher has interrogated all the PRQBICKs on WAIT_LIST,[7] control returns to the Initialization routine which then invokes the Message-Sender. The Message-Sender examines the MSGBLOKs on SEND_LIST. The TO_ID of the MSGBLOK is compared to the ID in each of the Line Control Blocks to find the TP line to be used to send the message to the proper computer. If the flags in the line block indicate the link is currently complete (the line has "signed on") and no messages are currently active on the line, the TO_ID, FROM_ID, MSG_ID, USER_ID, and message are taken from the MSGBLOK and placed in the Line Control Block output buffer in the proper format

---

[7] This may require two passes over WAIT_LIST if a process sends a message to a process on the same computer. This is because a new MSGBLOK will have been added to RECV_LIST. This MSGBLOK will contain a message that a previously examined PRQBLOK is waiting to retrieve.

for transmitting[8] and then sent (by making a request to a TP Monitor, for example). The MSGBLCK is not deleted from SEND_IIST at this time; instead a pointer to it is placed in the line block indicating it is the message currently active on the line. This is necessary because the MSGBLOK can't be deleted just yet since an acknowledgement has not been received from the MCCS receiving the message indicating it was correctly received. When a positive acknowledgement is eventually received, the Message-Receiver may then safely delete the MSGBLOK from SEND_LIST using the pointer obtained from the line block.

The remaining MSGBLOKs on SEND_LIST are examined in a similar manner with as many being sent as possible. When they have all been processed, control returns to the Initialization routine and it then waits for the SEND, RECV, or Message-Receiver routines to add something to WAIT_LIST, SEND_LIST, or RECV_LIST. When this happens, the Process-Dispatcher and Message-Sender are called again as described above.

The SEND and RECV routines and Message-Reviever are called "interrupt driven" routines in that they are activated at arbitrary times rather than sequentially like the Process-Dispatcher and Message-Sender.

The SEND and RECV routines are activated any time a process wishes to send or receive a message. As mentioned above, they allocate a buffer in which a PRQBLOK is built.[9] The PRQBLOK is then added to the end of WAIT_LIST so that it can be examined by the Process-Dispatcher. SEND and RECV then somehow suspend the process, either by suspending themselves (provided they are reentrant of course) or by placing the process's state in the PRQBLOK, marking the process non-dispatchable, and exiting to the operating system's dispatcher to allow another system task to run. Another function of SEND and RECV is to notify the Initialization routine that it is time to make another iteration through the loop which calls the Process-Dispatcher and the Message-Sender.

The Message-Receiver is entered when a message is received on the TP line (usually by accepting an interrupt). If the

---

[8] The format depends on whether the asynchronous or binary synchronous protocol is being used on the line. These formats will be precisely defined later.

[9] If there is no buffer space to be allocated, they immediately suspend themselves and, consequently, the process that called them, until such time that buffer space becomes available.

message that is received serves to acknowledge the last message sent on that line,[10] the MSGBLOK for that message (pointed to by the Line Control Block) is removed from SEND_LIST and the buffer freed.

Negative acknowledgements (or an undecodable message if there is an unacknowledged message active on the line) mean the active message must be resent. The Message-Receiver simply informs the Initialization routine to make another pass through its loop and the Message-Sender will resend the message since it was never removed from SEND_LIST.

A message that is received from a process on another computer is placed in a MSGBLOK and added to RECV_LIST if buffer space is available for creating the MSGBLOK.[11]

A postive acknowledgement is then constructed for the message just received. It may then be immedately sent back or added to the front of SEND_LIST so that it will be the next message sent on that line by the Message-Sender.

When there is no buffer space available to allocate a buffer for the MSGBLOK or an undecodable message is received and no message is active (sent but not acknowledged), a negative acknowledgement message is constructed and handled as the positive acknowledgement described above.

Again an extension is easily added to MCCS at this point.[12] A limited "store and forward" function may be provided by adding the MSGBLOK to SEND_LIST if it is not for a process on this computer but is for one on a computer which MCCS has a link to. (This is determined by comparing TO_ID with the IDs in all the line blocks.) If routing tables are also added, messages may be sent to computers to

---

10 The format of the acknowledgement message is defined in the section on aysnchronous and binary synchronous line protocols.

11 A special case must be checked for and handled if detected at this point. If a message is received and a positive acknowledgement returned but this acknowledgement is lost or garbled, the sending MCCS will resend the last message. Therefore the FROM_ID and the MSG_ID of the last message received on each line must be maintained in the Line Control Block. Duplicates can then be detected and simply acknowledged, but not added to RECV_LIST since there is already one copy there.

12 This extension has been implemented in the version of MCCS written to run on the IBM 370 (described in Part III).

which there is no direct link. This is accomplished by sending it to a computer to which there is a direct link, and it has, or can achieve through additional routing, a link to the desired computer.

Algorithmic Description.

The following PL/I type algorithm will serve to illustrate at a high level of abstraction the functions and data structure manipulation performed by MCCS. Several liberties have been taken to simplify the algorithm. For example, it is assumed that there are two primitives, add and remove, that perform additions and deletions to linked lists. Also, the assumption is made that the "P" and "V" primitives as described by Dijkstra [DIJ65] are available. An additional primitive, "CP", is used and is an extension to "P" that causes a conditional P to be performed. That is, if the semaphore used with CP is positive, a normal P is performed and a return code set to zero. If the semaphore is not positive, the CP only sets the return code to 1.

```
Initialization:procedure;
dcl 1 MSGBLOK based MSGPTR,   /*control blocks contained*/
    2 NEXT ptr,               /*on SEND_LIST and RECV_LIST*/
    2 TO_ID char,
    2 FROM_ID char,
    2 MSG_ID integer,
    2 USER_ID integer,
    2 MESSAGE char;

dcl 1 PRQBLOK based PRQPTR,   /*control blocks contained*/
    2 NEXT ptr,               /*on WAIT_LIST*/
    2 TO_ID char,
    2 FROM_ID char,
    2 MSG_ID integer,
    2 USER_ID integer,
    2 MESSAGE char,
    2 STATUS char,
    2 OP char,
    2 SEM semaphore;


dcl 1 LNEBLOK based LNEPTR,   /*Line Control Blocks*/
    2 NEXT ptr,
    2 ID char,
    2 STATE flag,
    2 MSGPTR ptr,
    2 LAST_FROM_ID char,
    2 LAST_MSG_ID integer,
    2 DEVICE number;
```

```
/*pointers to head of WAIT/SEND/RECV_LIST*/
dcl WAIT_LIST, SEND_LIST, RECV_LIST ptr;

/*semaphores to gain exclusive access to above lists*/
dcl WAIT_LOCK, SEND_LOCK, RECV_LOCK semaphore;

/*semaphore to cause Process-Dispatcher and Message-*/
/*Sender to be invoked again by Initialization routine*/
dcl WORK_TO_DO semaphore;

/*semaphore to "allocate" and "free" buffers*/
dcl BUFF_SEM semaphore;

/*counter to generate unique MSG_IDs*/
dcl MSG_COUNTER integer;

dcl ACK, NAK char;
  .
  .
  .
WAIT_LIST <- null;
SEND_LIST <- null;
RECV_LIST <- null;
MSG_COUNTER <- 0;
WAIT_LOCK <- 1;
SEND_LOCK <- 1;
RECV_LOCK <- 1;
WORK_TO_DO <- 0;
BUFF_SEM <- "number of available buffers";
ACK <- "positive acknowledgement";
NAK <- "negative acknowledgement";
  .
  .
  other initialization such as checkpoint recovery.
  .
  .
do forever;
   P(WORK_TO_DO);  /*wait until something to do*/
   call Process-Dispatcher;
   call Message-Sender;
end;
end Initialization;
```

```
Process-Dispatcher:procedure;
P(WAIT_LOCK);
PRQPTR <- WAIT_LIST;
do while PRQPTR ¬= null;
   if PRQBLOK.OP = 'SEND' then
      do;
         find LNEPTR suchthat LNEBLOK.ID = PRQBLOK.TO_ID;
         if LNEPTR = null then  /*invalid TO_ID*/
            do;
               PRQBLOK.FROM_ID <- blanks;
               V(PRQBLOK.SEM);
               goto next;  /*done with this PRQBLOK*/
            end;
         else;
         CP(BUFF_SEM);
         if return_code = zero then
            do;
               MSG_COUNTER <- MSG_COUNTER+1;
               PRQBLOK.MSG_ID <- MSG_COUNTER;
               if PRQBLOK.USERID = zero then
                  PRQBLOK.USERID <- MSG_COUNTER;
               MSGPTR <- addr buffer;
               MSGBLOK <- PRQBLOK, by name;
               P(SEND_LOCK);
               add MSGBLOK to SEND_LIST;
               V(SEND_LOCK);
               V(PRQBLOK.SEM);
            end;
         else;  /*leave process suspended*/
      end;
```

```
   else do;  /*must be request to RECV*/
      find LNEPTR suchthat LNEBLOK.ID = MSGBLOK.FROM_ID;
      if LNEPTR = null then  /*invalid FROM_ID*/
         goto resume;  /*go restart process*/
      P(RECV_LOCK);
      MSGPTR <- RECV_LIST;
      do while MSGPTR ¬= null;
         if PRQBLOK.TO_ID = MSGBLOK.TO_ID then
            if PRQBLOK.FROM_ID = MSGBLOK.FROM_ID or
                                 blanks then
               if PRQBLOK.USER_ID = MSGBLOK.USER_ID or
                                    zero then
                  do;
                     PRQBLOK <- MSGBLOK, by name;
                     remove MSGBLOK from RECV_LIST;
                     V(BUFF_SEM);
                     V(RECV_LOCK);
                     V(PRQBLOK.SEM);
                     goto next;
                  end;
               else;  /*wrong message*/
            else;  /*wrong message*/
         else;  /*wrong process*/
         MSGPTR <- MSGBLOK.NEXT;
      end;  /*search all MSGBLOKs*/
      V(RECV_LOCK);
      if PRQBLOK.STATUS ¬= zero then  /*won't wait*/
         do;
resume:     PRQBLOK.FROM_ID <- blanks;
            V(PRQBLOK.SEM);
         end;
      else;  /*leave process suspended*/
   end;
next:;
   PRQPTR <- PRQBLOK.NEXT;
end;
V(WAIT_LOCK);
end Process-Dispatcher;
```

```
Message-Sender:procedure;
P(SEND_LOCK);
MSGPTR <- SEND_LIST;
do while MSGPTR ¬= null;
   find LNEPTR suchthat LNEBLOK.ID = MSGBLOK.TO_ID
   if LNEBLOK.STATE = available then
      do;
         write formatted MSGBLOK to LNEBLOK.DEVICE;
         LNEBLOK.STATUS <- awaiting_response;
         LNEBLOK.MSGPTR <- MSGPTR;
      end;
   else;
   MSGPTR <- MSGBLOK.NEXT;
end;
V(SEND_LOCK);
end Message-Sender;
```

```
SEND:procedure(TO_ID,MESSAGE,MSG_ID,USER_ID) reentrant;
P(BUFF_SEM);  /*suspend ourself if no buffers*/
PRQPTR <- addr buffer;  /*get address of available buffer*/
PRQBLCK.TO_ID <- TO_ID;  /*build PRQBLOK from parms*/
PRQBLCK.FROM_ID <- "ID of calling process";
PRQBLOK.MESSAGE <- MESSAGE;
PRQBLCK.USER_ID <- USER_ID;
PRQBLOK.OP <- 'SEND';
PRQBLCK.SEM <- 0;
P(WAIT_LOCK);
add PRQBLCK to WAIT_LIST;
V(WAIT_LOCK);
V(WORK_TO_DO);
P(PRQBLOK.SEM);  /*wait until PRQBLOK acted upon*/
P(WAIT_LOCK);
remove PRQBLOK from WAIT_LIST;
V(WAIT_LOCK);
V(BUFF_SEM);
end SEND;
```

```
RECV:procedure(FROM_ID,MESSAGE,MSG_ID,USER_ID,STATUS)
                                                reentrant;
P(BUFF_SEM);  /*reserve buffer for MSGBLOK*/
PRQPTR <- addr buffer;  /*get address of available buffer*/
PRQBLCK.TC_ID <- "ID of calling process";
PRQBLOK.FROM_ID <- FROM_ID;
PRQBICK.USER_ID <- USER_ID;
PRQBLOK.STATUS <- STATUS;
PRQBICK.CF <- 'RECV';
PRQBLOK.SEM <- 0;
P(WAIT_LOCK);
add PRQBLOK to WAIT_LIST;
V(WAIT_LOCK);
V(WORK_TO_DO);
P(PRQBLOK.SEM);  /*wait until PRQBLOK acted upon*/
P(WAIT_LOCK);
remove PRQBLOK from WAIT_LIST;
V(WAIT_LOCK);
V(BUFF.SEM);
end RECV;



Message-Receiver:procedure;
find INEPTR for "interrupting TP line";
if "I/C error reading message" or
   "block check sum incorrect" then
   do;
      if LNEBLOK.STATUS=awaiting_response then
         do;
            LNEBLOK.STATUS <- available; /*free TP line*/
            LNEBLOK.MSGPTR <- null;  /*so message resent*/
            V(WORK_TO_DO);  /*when the Message-*/
            return;  /*Sender is called again*/
         end;
      else do;
         write NAK to LNEBLOK.DEVICE;
         return;
      end;
   end;
```

```
else do;
   if "input buffer" = NAK or ACK then
      do;
         if LNEBLOK.STATUS ¬= awaiting_response then return;
         if "input buffer" = NAK then  /*resend*/
            do;
               LNEBLOK.STATUS <- available;  /*free line*/
               LNEBLOK.MSGPTR <- null;  /*for use by the*/
               V(WORK_TO_DO);  /*Message-Sender*/
               return;  /*so message can be resent*/
            end;
         else do;  /*message was accepted so release it*/
            MSGPTR <- LNEBLOK.MSGPTR;  /*address of MSGBLOK*/
            P(SEND_LOCK);
            remove MSGBLOK from SEND_LIST;
            V(SEND_LOCK);
            V(BUFF_SEM);  /*make buffer available again*/
            LNEBLOK.STATUS <- available;  /*release line*/
            LNEBLOK.MSGPTR <- null;
            V(WORK_TO_DO);  /*may be more messages to send*/
            return;
         end;

      end;
   else do;  /*we have received a message*/
      CP(BUFF_SEM);  /*is there a buffer to have*/
      if return_code = zero then
         do;
            MSGPTR <- addr buffer;
            MSGBLOK <- "reconstructed input buffer";
            P(RECV_LOCK);
            add MSGBLOK to RECV_LIST;
            V(RECV_LOCK);
            write ACK to LNEBLOK.DEVICE;
            V(WORK_TO_DO);  /*may be a PRQBLOK waiting*/
            return;
         end;
      else do;  /*don't have buffer for the message*/
         write NAK to LNEBLOK.DEVICE;  /*ask to resend*/
         return;  /*maybe we will have buffers then*/
      end;
   end;
end;
end Message-Receiver;
```

Message_Format.

The format of the messages that are transmitted between computers by MCCS is shown in Figure 5. Although certain control characters must be added and reformatting done based on the line protocol being used (discussed in the next two sections), this format serves as the basis for all MCCS message communication. In the following definition of the fields in MCCS messages, "sending process" means the process that called SEND to send a message, "receiving process" is the process that the message was sent to, and "sending MCCS" is the MCCS that handles the request to send the message and subsequently transmits the message to the "receiving MCCS" which accepts the message and makes it available for retrieval by the receiving process. The parameters and their functions are as follows:

TO_ID and FROM_ID are the six character (in ASCII code) identifiers that specify the computer and process the message was sent to and from, respectively. The last four characters of the TO_ID and FROM_ID (the process-id) must contain at least one non-blank character when sending messages between processes. This is necessary because a TO_ID or FROM_ID ending in four blanks is used when MCCS on one computer needs to send a message to MCCS on another computer.

MSG_ID is a 16 bit integer that uniquely identifies the message within one MCCS. (The messages are sequentially numbered with the problem associated with MSG_ID overflowing and no longer being unique ignored since it happens only once every 2**16 - 1 messages.)

USER_ID is a 16 bit field assigned by the process which sent the message by using the USER_ID parameter in the call to SEND.

FLAG1 is a 8 bit field for internal use by the MCCS sending the message. It is ignored by the MCCS that receives the message.

FLAG2 is a 8 bit field used for communications between the sending and receiving MCCSs. Figure 6 gives a further breakdown of this field. F, when "1", indicates this is a "fake" message and is used when one of the other FLAG2 bits must be set and no "real" message is available to send. W is used with BSC protocol and, when "1", means that the MCCS sending this message cannot accept another message for a while (no buffer space for example).

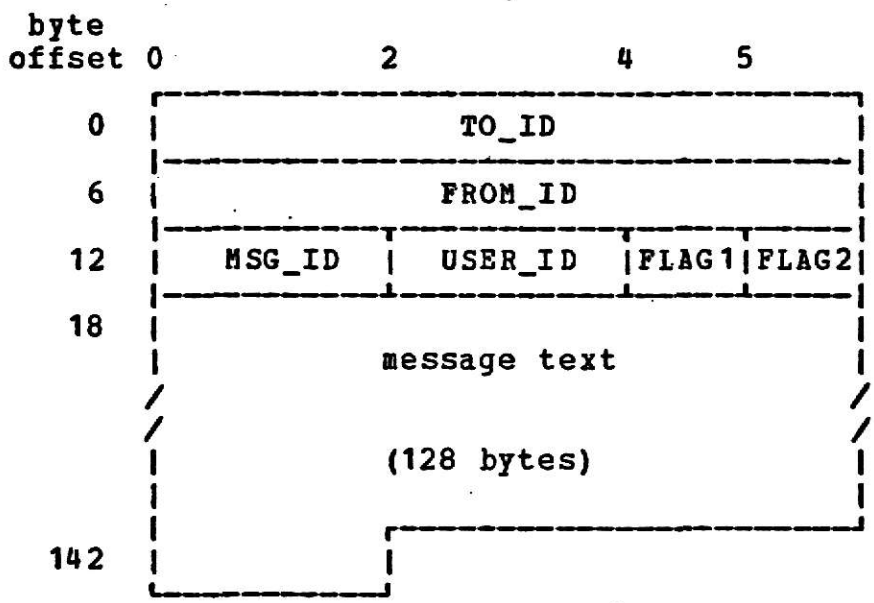


Figure 5: Message Format

Figure 6: FLAG2 Bit Definitions

When the receiving MCCS gets a message with the W bit on, then it should not send a message in response. D, when set to "1", indicates that the sending MCCS wishes to break the communications connection (disconnect). The receiving MCCS will send back a positive acknowledgement and disable the TP line on its end. When the sending MCCS receives the positive acknowledgement it disables the TP line on its end and the disconnect sequence is complete. U is currently unassigned. The four bits that make up "seq#" are currently not used but should be reserved for BSC sequence numbers messages if full duplex transmission is supported in the future.

The message text is 128 bytes long and its format and content are not examined by MCCS, i.e., it is

treated as binary data. No translations from one character code to another (ASCII to EBCDIC for example) are performed when transmitting the message from one computer to another. This is necessary because MCCS does not know which parts of the message are "characters" (they can be translated) and which parts are binary data (they cannot). If any translation needs to be performed, it must be done by the sending or receiving process with prior agreement with the other process. Processes wishing to sent information in lengths of other than 128 bytes should reformat the information into 128 byte "blocks" and include sufficient control information with the message text to allow it to be reconstructed by the recipient process.

Asynchronous Line Protocol.

This section defines the protocol used by MCCS when using asynchronous (ASC) TP lines. MCCS ASC protocol uses TWX line code and lies within the protocol defined for communication with Common Carrier TWX Terminals Model 33/35 (see section entitled "Telegraph Terminal Control Type II" in [IBM04] for technical specifications). This allows existing hardware and software that support teletype-like devices to be used in the implementation of MCCS.

Since several characters[13] have special line control functions, all occurrences of these characters must be removed from the message (Figure 5) before it is sent. This is referred to as making the message transparent. It is done by replacing each occurrence of one of these characters by the DLE (X'10') character and immediately followed by an encoded non-control version of the character. (Note that this now makes DLE a control character.) Table 1 gives the replacement characters for each control character. MCCS also computes a very simplified version of a Block Check Sum (BCC) and appends it to the end of the message (the BCC must also be made transparent). The BCC is computed as the algebraic sum of all the individual characters in the message (after the replacements noted above have been made) and stored as a 16 bit integer following the last byte of the message. Finally, a carriage return (X'0D' or X'8D') is added after the BCC. The message that is sent, therefore,

---

[13] These characters are: X'04', X'05', X'0D', X'7F', X'84', X'85', X'8D', X'DF', and X'FF'.

CHAR	REPLACEMENT	CHAR	REPLACEMENT
X'04'	X'1011'	X'84'	X'1015'
X'05'	X'1012'	X'85'	X'1016'
X'0D'	X'1013'	X'8D'	X'1017'
X'10'	X'1010'	X'DF'	X'1018'
X'7F'	X'1014'	X'FF'	X'1019'

Table 1: Control Character Replacement

may be from 147 to 293 bytes long.[14] The transparent message is then converted to TWX line code (the bits within each byte are reversed: B'10101010' becomes B'01010101', etc.) and transmitted on the TP line. The receiving MCCS then goes through a similar process, only in reverse order, to convert the message that was received back to the form shown in Figure 5. A new BCC is computed when the message is being reconstructed and compared with the BCC that was received to determine if any line errors occurred.

The normal state for MCCS in relation to an ASC line is "ready to receive". This means MCCS is ready to accept a message from the MCCS on the computer connected to the other end of the TP line. When a message is received, it is re-formatted as noted above. Several checks are then made to verify the message was received correctly: the BCC is recomputed and verified, and the TO_ID and FROM_ID fields are checked (refer to the previous discussion of the Message-Sender). Either of these conditions will cause a negative acknowledgement to be returned, indicating the message should be resent.

An acknowledgement message is then constructed so that it may be returned to the sending MCCS to indicate the message was received correctly or should be resent. An acknowledgement message is just a special case of the general message shown in Figure 5. The TO_ID is set to the two character ID (and padded with four blanks) of the

---

[14] The message takes 144 bytes, the BCC two bytes, and the carriage return one byte. In the worst case (when all the characters in the message and BCC are control characters) the message takes 288 bytes, the BCC four bytes (if both bytes of the BCC form control characters), and one byte for the carriage return.

computer which sent the message and the FRCM_ID is set to the two character ID (again padded with blanks) of the computer which received the message. The MSG_ID is set as with any other message and the USER_ID is set to zero. The first bit of the message text is used to indicate whether this acknowledgement indicates the message was received correctly or should be resent. If this bit is set to "0", the message was received correctly, if it is "1", the message should be resent.

When MCCS has a message to send, the message is formatted as described above and transmitted on the appropriate TP line. If any I/O errors are detected when transmitting the message, up to 16 attempts are made to re-transmit. 16 consecutive I/O errors are considered to indicate some type of permanent error condition and the line is disabled and the communication link broken with that computer.

Once the message has been sent, MCCS prepares to receive the acknowledgement for the message. It is possible under some unusual circumstances for the message to appear to have been transmitted correctly, mostly due to abnormal conditions on the TP lines, yet it is never received by the receiving MCCS. Therefore it is necessary to have some maximum time that MCCS will wait to receive the acknowledgement message. This time is somewhat arbitrary and should normally be in the range of five to 30 seconds, depending on the expected response time for the computers and TP lines involved. If no acknowledgement is received in this time, the message is resent up to 16 times before the condition is considered unrecoverable.

When the acknowledgement is received, the first bit of the message text is checked to see if the message was received correctly or should be resent. If the bit is "1", the message is resent (again, 16 retries are made before considering the error permanent). Otherwise the message was received correctly so it may be removed from SEND_LIST and another message formatted for transmission (if one exists).

It is possible that a message will be received while waiting for an acknowledgement. If this happens, the message is accepted and acknowledged as normal and the message that was originally sent is then resent.

Binary_Synchronous_Line_Protocol.

MCCS BSC protocol provides for a very efficient form of communication which utilizes high-speed BSC TP lines with a high level of transmission error detection and recovery. (The reader unfamiliar with general BSC terminology and protcccl is referred to [IBM01] as this topic will not be discussed here.)

MCCS BSC protocol is based on that used by IBM's HASP MULTI-LEAVING.[IBM06] Unlike the ASC protocol described above which uses a contention[15] system, MCCSs using BSC protocol alternate sending a message (or an acknowledgement message if there is no message available) and awaiting the response. HASP calls this "handshaking" since the two computers are always kept in synchronization, both knowing which has the line for transmitting and which for receiving.

There are three basic data formats that are transmitted with MCCS BSC protocol.[16] The first of these formats is used to transmit the message shown in Figure 5. Since there may be some characters within the message that have BSC control functions, BSC transparent-text mode is used in transmitting all MCCS messages. The format of the message, when transmitted on BSC lines, is then:

DLE,STX,message,DLE,ETX,block-check-sum

with all occurrences of DLE (X'10') within "message" being replaced by DLE,DLE and with synchronization characters (DLE,SYN) added as as needed (described in [IBM01]) to establish and maintain bit and character phase. Compression of the message is not currently defined for MCCS.

The second format is the positive acknowledgement sequence, ACK0 (SYN,SYN,DLE,X'70' including sync and pad characters). ACK0 is used to acknowledge the correct receipt of the last

---

15 Contention means that either computer may arbitrarily try tc transmit a message, with the possibility that both may request the line at the same time. Presumably the simultaneous request can be resolved by repeated retries of transmitting the message.

16 The control characters, synchronization and pad characters, and block check sums used by MCCS conform to those defined in [IBM01] and for the "Synchronous Data Adapter - Type II" in [IBM03] for EBCDIC transmission code. Since these control sequences are required for all BSC transmission, they will not be shown in the description of MCCS BSC protocol.

message sent when there is no message available to transmit back to serve this purpose.

The third and final format is that of the negative acknowledgement, NAK (SYN,SYN,NAK,X'F0' including sync and pad characters). NAK is used to indicate the message just received was not received correctly due to I/O errors reading the message or an incorrect block check sum and that it consequently should be retransmitted. NAK is also sent if no message is received within a predefined timeout period (usually 30 seconds to one minute). The timeout avoids the deadlock situation where one MCCS sends a message which is completly lost during transmission and both MCCSs are relegated to waiting for a message and response that will never appear.

Once the communications link has been established between two MCCSs, they simply take turns sending messages, if any are available, transmitting ACK0s otherwise, and waiting for the response. Before this handshaking can commence, however, one MCCS must send the first message and the other MCCS wait until this message is received before actual message transmission can begin. For this purpose, one MCCS is designated as a "host" and the other a "remote". (With dial-up communication lines, the MCCS on the end that actually "dials" is considered to be the remote. This causes the host to wait until such time that the MCCS dialing in indicates it is ready for transmission. When leased communications lines are used, the host-remote designation is somewhat arbitrary. In any case, an implementation of MCCS should support BSC links in both host and remote mode, preferably being changeable by operator command.) The remote MCCS then sends the first message when it is ready to begin communication. Likewise, the host waits until this first message is received (timeouts are not applicable here) before sending any messages destined for the remote MCCS. Other than in determining which MCCS sends the first message, a host and a remote MCCS are identical.

As mentioned above, two MCCSs carry on a "conversation" which consists of sending a message and awaiting a response. To avoid unnecessary line turn-arounds and delays, a message destined for the MCCS which sent the message just received may be returned as a postive acknowledgement for that message. If each MCCS has messages for the other, no "unnecessary" data therefore need be transmitted. If a MCCS does not have a message to return, then the ACK0 must be transmitted to serve as the acknowledgement. (A NAK will of course be returned if the message should be resent.) If neither MCCS has messages to send, they simply exchange ACK0s to maintain proper synchronization. This can easily consume a considerable amount of both CPU and I/O resource if an ACK0 is immediatly returned in response to a ACK0.

Therefore, if a MCCS receives a ACK0 and has nothing to return but ACK0, it will wait for 2 seconds. At the end of 2 seconds, a message is transmitted if one has become available, otherwise the ACK0 is sent. This allows for the two MCCSs to maintain synchronization yet greatly reduces the overhead involved.

The section entitled "Message Format" described a bit within the FIAG2 field of the message which has special meaning in BSC transmission. This is the "W" bit. When it is one, the MCCS which sent the message is not capable of accepting a message in response (unless it is a "fake" message). The MCCS which receives a message with this bit on should then wait 2 seconds and respond with an ACK0 (or "fake" message). If the next message received has this bit on again, MCCS again waits two seconds and responds with a ACK0. This process is repeated until a message with the bit off, ACK0, or NAK is received at which time normal communication may resume.

Future Enhancements.

Several useful extensions were mentioned in the preceding definition of MCCS. Two of them, allowing processes to send and receive messages from a process on the same computer and one level of indirection in message routing should be included in all but the most basic MCCS implementations. Two others, multiple levels of indirection in message routing and message compression in BSC transmission are not so trivial.

Multiple level message routing requires that a MCCS know which computer to relay a message to when it does not have a direct link to the computer which contains the recipient process. It therefore must know the computer it relays the message to has such a direct link or knows of yet another computer which does, and so on, until the message finally arrives at its destination. This may obviously be done by each MCCS having a set of "routing" tables which indicate which computer to relay a message to when there is no direct link available. The problem arises in building and maintaining this table. If the network of MCCSs is static, the tables may be established once and left unchanged. This is generally unacceptable however, since the network will be dynamically changing or one of the computers may be inoperative and an alternate path needed to bypass it. This means that the MCCSs must be capable of altering the routing table dynamically by communicating changes in the network configuration. There is currently no facility for this (except that "inter-MCCS" messages are supported by having

the last four characters of the TO_ID and FROM_ID fields being blank), although its merits are worthy of additional evaluation to determine if it should be added at some later time.

A second difficult extension, compression of BSC messages, causes the transmission time required to send a message to be reduced by cutting down on the number of characters that are actually sent. There are several algorithms [IBM06] for doing this, although the basic idea is to replace multiple consecutive occurrences of a particular character with a single occurrence of that character and a count of the number of times it occurred in the original message. Control information is then placed at the beginning of the message to indicate where these character-count fields are located. If a majority of the messages being transmitted contain repetitive character sequences, compression may save a considerable amount of transmission time. Again, there is currently no facility in MCCS to do this, however it is worth further investigation.

There are obviously many more extensions that can be made to MCCS. Supporting full BSC transmission, adding additional line protocols, allowing varing length messages, and more sophisticated user-level primitives are all possibilities. But one must not lose sight of the original MCCS objective stated in the Introduction: That MCCS should remain simple enough that it may be implemented on nearly any computer. If too many extensions are added, the system becomes so complex that this objective is no longer obtainable.

III MCCS/370 - THE VM/370 IMPLEMENTATION OF MCCS

MCCS/370 is the name given to the set of routines, written in 370 assembler language, which comprise an implementation of MCCS on an IBM System/370 Model 158 running VM/370 with CMS.[17] This part then describes that implementation.

Since it takes a great deal of effort and space to describe all the IBM hardware and software components utilized by MCCS/370 (evidenced by the enormous number of IBM manuals that do this), no attempt is made to duplicate this information here.[18] References to specific IBM manuals will be given when appropriate, however a general knowledge of IBM terminology, the 370 instruction set and I/O, and the CP and CMS command language is almost a prerequisite for reading this section. The reader that is unfamiliar with the 370/158, the 3705 communications controller, 1052 consoles, or CP and CMS is therefore advised to at least review the appropriate introductory IBM manual listed in the Bibliography.

Operating Environment.

MCCS/370 is designed to run stand-alone in a virtual machine simulated by CP. This machine must have at least 36K[19] of memory, a virtual console, a virtual printer, a virtual card reader at device address X'00C', and a virtual card punch at X'00D'. Additionally, virtual asynchronous TP lines will be defined by MCCS as needed. "Real" BSC TP lines must be attached to the MCCS virtual machine by the VM system operator since they are not simulated by CP.

Because MCCS/370 runs stand-alone in its own virtual machine, not under the control of another operating system, the processes sending and receiving messages must execute in another virtual machine. Part of the MCCS/370 implementation therefore includes the routines to provide a means of communication between the MCCS virtual machine and the virtual machines executing these processes. This

---

17 The implementation and testing of MCCS/370 was done under Release 2 PLC 13 of both CP and CMS.

18 Not to mention the problems associatated with keeping duplicate documentation appearing in many places updated concurrently.

19 This memory is used to hold the MCCS code, any additional memory is used to provide buffer space for messages.

communication is accomplished by using virtual Channel To Channel Adapters (CTCAs) to move data between the virtual machines.

Routines to implement the SEND and RECV primitives have been written to allow processes running under CMS to send and receive messages using MCCS. These routines dynamically define a CTCA, couple it to MCCS/370s CTCA, and transfer the appropriate information to cause the send or receive request to be performed. The CTCA is then detached and the process continues execution.

MCCS/370 is designed so that it may run disconnected, that is, with no real operator's console and, therefore, no operator intervention. Any unexpected program interrupts or unrecoverable error conditions during MCCS execution are detected, a dump of the virtual machines' storage taken (for later problem analysis), and MCCS is restarted.

Checkpointing is accomplished by writing the information contained in WAITLIST, SENDLIST, and RECVLIST (the MCCS/370 names for the WAIT_LIST, SEND_LIST, and RECV_LIST of Part II) to the virtual card punch (which is SPOOLed to "*") each time they change. Part of MCCS initialization therefore involves examining the spool files in the virtual reader for checkpoint data, and recovering it if it exists.

MCCS/370 is started from CMS by using the VM/370 procedures for generating a CP or CMS system. [IBM12] [IBM10] The VMFLOAD routine is used to punch the IPL-able loader, DMKLD00E, and the TEXT decks which make up MCCS/370. The virtual card reader is then IPLed causing DMKLD00E to be loaded which in turn loads MCCS and passes control to the MCCS virtual machine initialization routine.

When MCCS/370 must restart itself after a program failure, an IPL command is executed which causes CMS to be loaded and then VMFLOAD invoked to start the MCCS loading procedure as described above.

Flow of Control.

The flow of control in MCCS/370 followes very closely that given in Part II for the general MCCS. Figure 7 shows the MCCS/370 routine names and execution flow. The data strutures in MCCS/370 also parallel those given in Part II. PRQBLOKs are contained on WAITLIST, MSGBLOKs on SENDLIST and RECVLIST, and LNEBLOKs (the MCCS/370 name for Line Control Blocks) describe the TP lines.[20] A very brief description of the function of each MCCS routine will be given at this

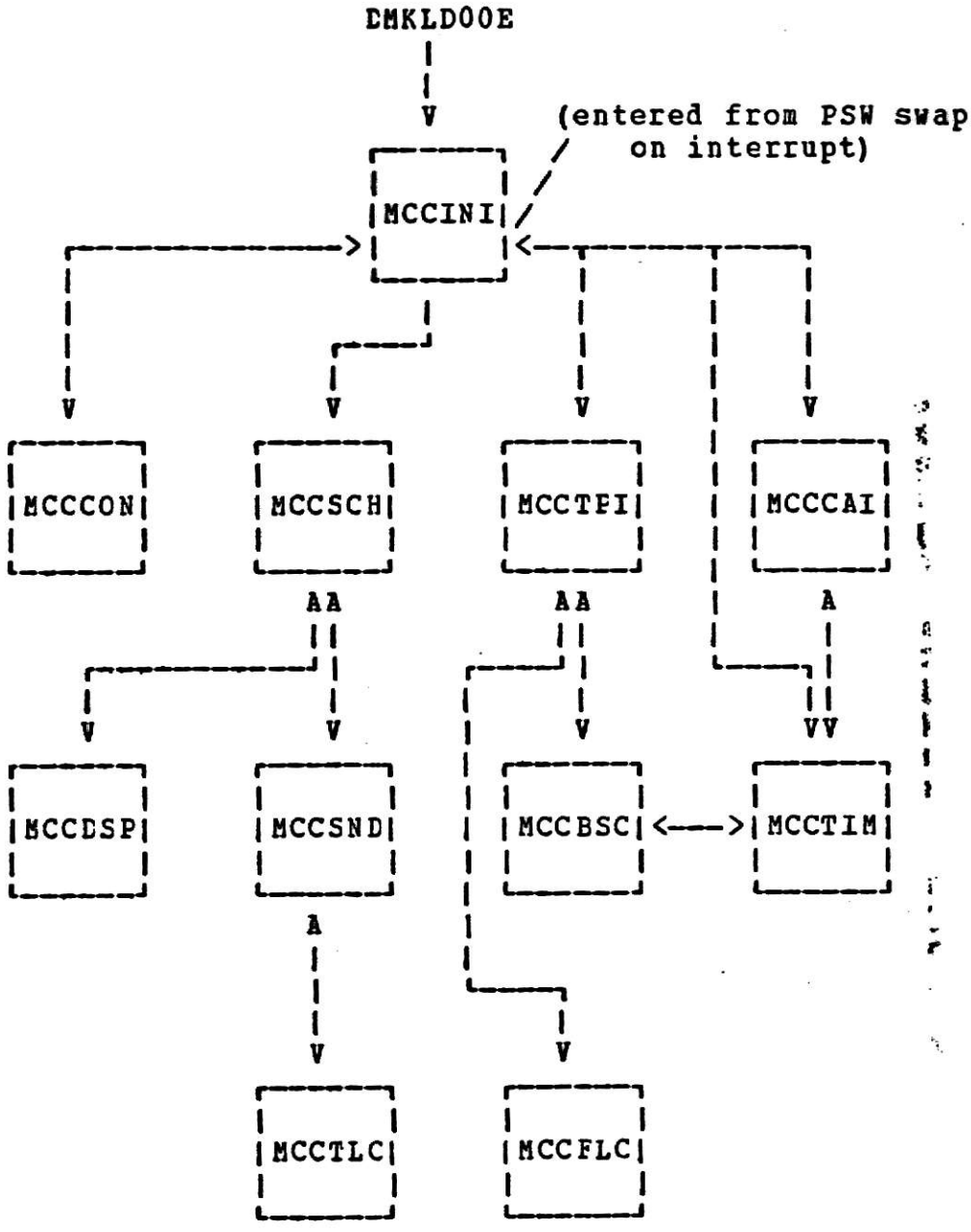


Figure 7: MCCS/370 Module Diagram[21]

---

20 These control blocks are listed in Appendix E (Page 81).

21 MCCCKP, the checkpoint routine, is not shown in order to simplify the diagram. It is called by MCCBSC, MCCCAI, MCCDSP, and MCCSND when they modify any of the data structures.

time. A more detailed discussion of the routines operation and interaction with other routines is given later. Appendix A (Page 57) contains a more technical synopsis of each routine (entry points, entry conditions, external references, register usage, etc.).

MCCBSC is the BSC line driver. It handles all interrupts for BSC lines, placing incoming messages on SENDLIST or RECVLIST, selecting messages from SENDLIST to be transmitted, and handling BSC I/O errors.

MCCCAI is the Channel to Channel Adapter interrupt handler. Attention interrupts on the channel indicate that a process in another virtual machine as issued a SEND or RECV request and the request description should be read from the channel. MCCCAI then reads this description and adds it to WAITLIST.

MCCCKP is the checkpoint routine. It is called when WAITLIST, SENDLIST, or RECVLIST has been changed and therefore needs to be re-checkpointed. The information is written to the virtual card punch by MCCCKP so that it may be recovered after a system failure.

MCCCCN is the virtual machine's console handler. All console interrupts cause this routine to be invoked. The operator command is read, scanned, and processed as described in the section entitled "Operator Commands".

MCCCWR executes in the virtual machine with the process that wishes to use MCCS/370. It performs all the Channel to Channel Adapter I/O needed to communicate the SEND or RECV request to the MCCS virtual machine.

MCCDSP is the Process-Dispatcher of Part II. It examines the PRQBLOKs on WAITLIST taking messages to be sent and placing the message on SENDLIST and notifing MCCCWR (via Channel to Channel Adapter I/O) that the process may continue. Requests to receive a message are handled in a similar manner by searching RECVLIST for the requested message and returning it (through MCCCWR) to the process if found.

MCCFLC is called to convert a message that has just been received on an asynchronous TP line back to the standard message format. This includes converting from TWX line code, removing transparent character sequences, and validating the block check sum.

MCCGSI is called to obtain the spool file id number of the spool file currently active on the virtual punch. This spool id is used by MCCCKP to purge an existing version of checkpoint data when a updated copy is created.

MCCEND is a routine that serves as an indication of the highest memory address occupied by MCCS routines. It contains no executable code. Because it is the last routine that is loaded during MCCS initialization, its address may be used to mark the first location available for buffer allocation.

MCCINI is the first MCCS routine to gain control after Initial Program Load (IPL). It performs the initialization necessary to make the virtual machine suitable for executing other MCCS routines. This includes such things as setting up the PSWs in the nucleus so that interrupts may be handled properly.

MCCRUM executes in the virtual machine with a process utilizing MCCS. It is invoked when a process executes a RECV request. It takes the process's parameters and places them in a PRQBLOK and calls MCCCWR to communicate the request to the MCCS virtual machine.

MCCSCH performs the MCCS related initialization. Any existing checkpoint data is used to build the initial contents of WAITLIST, SENDLIST, or RECVLIST. The logic to handle unrecoverable error situations (including MCCS restart) is contained in MCCSCH. The loop that repeatedly calls the Process-Dispatcher (MCCDSP) and the Message-Sender (MCCSND) when their services are required is also part of MCCSCH's function.

MCCSND is responsible for performing the duties assigned to the Message-Sender of Part II for messages that are transmitted on the asynchronous TP lines. Messages contained in the MSGBLOKs on SENDLIST are written to the appropriate TP line when it is available for transmission.

MCCSUM executes in the virtual machine with the process which utilizes MCCS/370. It is invoked when the process issues a request to SEND a message. MCCSUM operation parallels that of MCCRUM described above.

MCCTIM handles timer (both interval and time of day) interrupts for the MCCS virtual machine. MCCTIM also provides the capability for another MCCS routine to request that control be passed to a certain location

after a specified interval of time has elapsed.

MCCTLC is the counterpart of MCCFLC. It is called to prepare a message for transmission on an asynchronous TP line. The control characters in the message are replaced by the proper transparent sequence, the block check sum calculated, and the message converted to TWX line code.

MCCTPI is the interrupt handler and Message-Reciever for asynchronous TP lines. Incoming messages are added to RECVLIST (or SENDLIST if not for a process on the 370) and the proper response constructed and added to SENDLIST so that it will be returned by way of MCCSND.

MCCINI receives control from DMKLD00E after all the MCCS routines have been loaded into memory. This routine is responsible for all the initialization of the MCCS virtual machine plus performing the first level of interrupt handling.

MCCINI begins by setting the program new PSW to cause a disabled wait PSW to be loaded if a program check occurs during initialization (the right half of the PSW will contain ' PGM'). Virtual console spooling is then started so that a log may be kept of all MCCS activity. The virtual console address is then obtained from CP using DIAGNOSE X'0024' and stored in MCCSECT.[22] The MCCS identification message is then written to the console along with the current date and time.

The program new PSW is then temporarily replaced so that the MCCS virtual machine storage size may be obtained. This is done by entering a loop which does an ISK instruction for each 4K page until an addressing interrupt occurs or all 16M of memory has been examined. The calculated memory size is then stored at location FREEHI in the nucleus for use by the storage management routines and the program new PSW is restored. The address of routine MCCEND, a dummy routine which is the last routine loaded by DMKLD00E, is rounded up to the next highest page boundary and stored at location FREEHWM to indicate the start of free storage.

The restart, external, SVC, and I/O new PSWs are then set to

---

22 MCCSECT is a DSECT (CSECT in MCCINI) which contains information that is shared by all MCCS routines. This includes things like the pointers to WAITLIST, SENDLIST, and RECVLIST and all the LNEBLOKs. All routines keep the address of MCCSECT in register 9.

cause control to be passed to the appropriate location within MCCINI when one of these interrupts occurs. The machine-check new PSW is set to cause a disabled wait PSW to be loaded (the right half of which contains ' MCK') since a machine check should never occur in a virtual machine. The address of MCCSCH is then obtained and control transferred to it so the MCCS related initialization may be performed.

When a program restart PSW swap occurs, control passes to label RESTART in MCCINI. A SVC 255 is then issued with a ABEND code of "RESTRT" (see the definition of SVC 255 in MCCSCH) which causes a dump to be taken and MCCS to be restarted.

An external interrupt PSW swap causes control to be passed to label EXTERNAL in MCCINI. If the interrupt code in the external old PSW is X'0004' (caused by issuing the CP EXTERNAL command), the wait bit in the external old PSW is turned off (if on). This will cause MCCSCH to call MCCDSP (the Process-Dispatcher) and MCCSND (the Message-Sender) just as if one of the interrupt routines had "POSTED" MCCSCH (this is further explained in the description of MCCSCH which follows). An external interrupt code of X'0080' (the interval timer) or X'1004' (the time of day clock comparator) cause MCCTIM to be called to process it. All other external interrupts are ignored.

The PSW swap that happens when an SVC instruction is executed causes control to be passed to label SVC in MCCINI. If the SVC code in the SVC old PSW is 255, control is passed to entry-point SVC255 in MCCSCH. SVC 0 and SVC 1 are used to allocate and free storage, respectively, and are described below. Any other SVC code is considered invalid and, if detected, causes control to be passed to entry-point SVCXXX in MCCSCH for processing.

All requests to allocate and free storage manipulate fixed size blocks. This is possible because storage is allocated only to hold a MSGBLOK or PRQBLOK which are very close to the same size. A linked list is maintained of all the free blocks of storage so that a block may be removed from the list, its address returned in register 1, and the condition code set to zero when an SVC 0 is executed. If no buffer is on the free list a check is made to see if the highest address allocated so far (contained at location FREEHWM) is less than the virtual machine size. If it is, free storage is extended by one block and FREEHWM incremented accordingly. (This technique eliminates the requirement that all of the blocks in free storage be linked together during the virtual machine initialization.) If free storage cannot be expanded, control returns with the condition code set to 3. To free a block of allocated storage an SVC 1 instruction is executed with register 1 containing the block

address. The block is then linked to the front of the free list so it is available for later reallocation.

An I/O interrupt PSW swap causes control to be passed to label IO in MCCINI. The CSW is loaded in regesters 2 and 3 and the device address in register 4. If the interrupting device is a CTCA, MCCCAI is called. Interrupts from a TP lines cause MCCTPI to be called. Console interrupts are handled by calling MCCCON. Interrupts from any other device are ignored.

MCCSCH receives control from MCCINI after the virtual machine intitialization has been completed. MCCSCH is responsible for MCCS initialization (including checkpoint recovery), dumping virtual storage and restarting MCCS when an unrecoverable error is detected, and calling MCCDSP and MCCSND to process PRQBLOKs on WAITLIST and MSGBLOKs on SENDIIST and RECVLIST.

MCCS initialization begins with the program new PSW being set to cause MCCSCH to be entered at label PROGINTR when a program interrupt occurs. Then several CP commands are issued which set the interval timer to reflect real time and to define the virtual CTCAs used for communication with other virtual machines. A check is then made to see if there is a real console associated with the virtual console (using DIAGNOSE X'0024' [IBM12]). If there is, a message is printed informing the operator that MCCS commands may now be entered. An enabled wait PSW is then loaded to allow the operator a chance to enter commands before initialization continues.[23]

After the operator has entered the desired commands[24] (or if there was no real console) the pointers to WAITLIST, SENDLIST, and RECVLIST in MCCSECT are zeroed to indicate the lists are empty. If checkpointing was not disabled by the operator, the SFBLOKs [IBM08] of the spool files in the virtual reader are examined by using DIAGNOSE X'0014'.[IBM12] If any spool file containing checkpoint data is found, it is read and used to build the initial contents of WAITLIST, SENDLIST, and RECVLIST, as appropriate.

The LNEBLOKs are then processed one by one with each line that is not drained (set by the MCCS DRAIN command) being enabled. If the LNEBLOK indicates the line is asynchronous,

---

23 MCCSCH does not resume until a MCCS BEGIN command is executed to turn off the wait bit in the I/O old PSW.

24 These commands are described in the section entitled "Operator Commands" that follows.

a CP DEFINE command is issued to to define a virtual teletype line at the appropriate address.[25] A SIO is then issued to cause the line to be disabled and enabled (to reset it). If the enable is not successful an SVC 255 is executed with an ABEND code of "SCH001". BSC lines are enabled by a DISABLE, SETMODE, ENABLE channel program to reset the line and insure it is in ITB mode.[IBM03] If the BSC enable fails, a message is logged on the console (the BSC might not have been attached yet). The remaining LNEBLCKs are processed in a similar manner.

MCCSCH then enters a non-terminating loop which repeatedly calls MCCDSP and MCCSND. First an enabled wait PSW is loaded at label WAITLOOP. When one of the interrupt driven routines (MCCBSC, MCCCAI, or MCCTPI) adds a control block to WAITLIST, SENDLIST, or RECVLIST that needs to be processed by MCCDSP or MCCSND, they turn off the wait bit in the I/O old PSW and place a "1" at location POSTFLG in MCCSECT. (The wait bit is also turned off by entering the MCCS BEGIN command or by a X'0040' external interrupt.) When the I/O old PSW is subsequently loaded when the routine has finished its processing, MCCSCH then begins execution at label RETRY. POSTFLG is reset to zero and MCCDSP and MCCSND are called. Interrupts are then disabled (to prevent multiple access) and POSTFLG is examined. If it is still zero, control passes to label WAITLOOP to await more work. If it is not zero, it was posted while MCCDSP and MCCSND were executing so they must be called again. Interrupts are enabled again and a branch is made to label RETRY.

When a program interrupt occurs during MCCS execution, the program new PSW causes MCCSCH to be entered at label PRCGINTR. The registers at the time of the interrupt are stored at location GPRLOG in the nucleus and an ABEND code of "PRGxxx" is constructed. (xxx is replaced by the program interrupt code.) Control then passes to label DUMPMCC.

MCCINI's SVC handler, upon encountering an SVC code of 255, enters MCCSCH at entry-point SVC255. By convention, SVC255 is used as a "die" SVC. When a MCCS/370 routine encounters an unrecoverable error situation it executes an SVC 255. The six bytes immediately following the SVC instruction contain a code identifing the error. (This code is usually the fourth through sixth characters of the routine's name followed by a three digit number. See Appendix B on Page 74.) Control then passes to label DUMPMCC.

---

[25] The device address of the TP lines are specifed by two constants in MCCSECT, LOLINE and HILINE. The difference in these two constants plus 1 therefore determines the number of LNEBLOKs in MCCSECT.

MCCSCH is entered at entry-point SVCXXX by MCCINI's SVC handler when an invalid SVC code is encountered. An ABEND code of the form "SVCxxx" is constructed (xxx is the invalid SVC code) and control passes to label DUMPMCC.

When control reaches label DUMPMCC, the address of the instruction at which the error occurred is determined and it, along with the ABEND code are logged on the console. They are also placed in the title of a CP DUMP command used to dump virtual memory to the printer.

The following CP command is then executed to cause MCCS/370 to be restarted:

```
IPL CMS PARM EX MCCRS[26]
```

The EXEC named MCCRS then invokes VMFLOAD [IBM10] with a loadlist EXEC named MCCLOAD and a CNTRL file named MCCR10. A CP IPL 00C command is then executed to cause the system load deck created by VMFLOAD to be IPLed so that DMKLD00E may load the MCCS modules.[IBM10]

MCCDSP receives control from MCCSCH and examines the PRQBLCKs on WAITLIST. It functions in a manner identical to the Process-Dispatcher described in Part II except in how processes are suspended and restarted. A discussion of the MCCS/370 routines MCCSUM, MCCRUM, and MCCCWR will be necessary at this point to expain how process suspension works.

When a process (executing under CMS in another virtual machine) wishes to send a message, it issues a call to SEND (an alternate entry-point of MCCSUM which has been linked in with the process). The process's parameters are placed in a PRQBLCK with the FROM_ID being set to the USERID of the virtual machine MCCSUM is running in. The PRQBLOK address is then passed to routine MCCCWR via call. MCCCWR then defines a CTCA and couples it to MCCS's[27] CTCA at device address X'100' and writes the PRQBLOK to the CTCA.

The PRQBLOK contains the USERID of the virtual machine

---

26 The "EX MCCRS" parm requires a KSU Computing Center modification to CMS which causes the EXEC named MCCRS to be invoked after CMS has been IPLed.

27 The CP COUPLE command that MCCCWR uses must contain the USERID of the virtual machine to couple to. This USERID is currently "VMJH8" and must be changed if a different virtual machine is used to run MCCS/370.

MCCSUM is executing in as well as the address of the CTCA that MCCCWR defined so that MCCDSP can later return a response to MCCCWR. MCCCWR then waits for this response, thereby suspending the process.

The CTCA interrupt handler in MCCS/370 (MCCCAI) reads the PRQBLCK from the CTCA and places it on WAITLIST, sets POSTFIG to 1, and turns the wait bit off in the I/O old PSW (causing MCCSCH to call MCCDSP and MCCSND). Calls to RECV function in a similar manner except MCCRUM is initally called and MCCCWR couples to MCCS's CTCA at device address X'110'.

When MCCDSP needs to restart a process after its PRQBLOK has been acted upon, it couples the CTCA at address X'120' for responses to SEND or X'130' for RECV requests, to the USERID and CTCA address specified in the PRQBLOK. The updated PRQBLOK is then written to the CTCA which causes MCCCWR to receive a interrupt and read the PRQBLOK. It then returns control to MCCSUM or MCCRUM which modify the appropriate parameters and return to the process.

MCCSND is called by MCCSCH to process the MSGBLOKs on SENDLIST. It functions identically to the Message-Sender described in Part II for messages transmitted on asynchronous TP lines. The MSGBLOK is made transparent and the block check sum computed by calling routine MCCTLC. When the MSGBLOK has been formatted, MCCSND writes the message to the TP line using SIO and places the MSGBLCK address in the appropriate LNEBLOK. The line is flagged as being active and, if the message just sent was actually a message (as opposed to a acknowledgement message), as awaiting a response. The next MSGBLOK on SENDLIST is then examined.

When MCCSND encounters a MSGBLOK that is to be sent on a BSC line, it loads the address of the appropriate LNEBLOK in register 1 and calls entry-point MCCTIMI in module MCCTIM. This is necessary because MCCBSC, which sends and receives all messages on BSC lines, may have called MCCTIMS to stack a TRQBLCK to wait two seconds before the next data is written due to no messages being available for sending (see the description of MCCBSC that follows). After control returns from MCCTIMI, MCCSND examines the next MSBLOK on SENDLIST. After all MSQBLOKs have been processed, control returns to MCCSCH.

MCCTLC and MCCFLC are routines which prepare a MSGBLOK for sending or reconstruct it after receiving, respectively, when using asynchronous lines. They provide for proper encoding and decoding of control characters and computation of the block check sum.

MCCCKP is the MCCS/370 checkpoint routine. It is called with 'WAIT', 'SEND', or 'RECV' in register 1 when that list has been modified. If checkpointing has been turned off by operator command, MCCCKP simply returns. Otherwise the contents of specified list are written to the virtual punch. This is done by placing the address of the first control block on the list in the CAW and issuing a SIO to device X'00D'. The first eight bytes of each PRQBLCK or MSGBLOK is a WRITE CCW which has a data address and length which cause the information in the block to be written. The CCWs in all but the last control block specify command chaining and are followed by a TIC CCW which contains the address of the next control block (which also serves as the general linkage of PRQBLOKs and MSGBLOKs). Routine MCCGSI[28] is called to return the spool file id number of the spool file currently active on the punch and this id is stored in MCCSECT. The punch is then closed which causes the checkpoint spool file to be placed in the reader since the punched was spooled to "*" by MCCSCH during initialization. The previous checkpoint file for the list is obtained from MCCSECT and purged (if it exists) so only one copy of the checkpoint data is present. Control then returns to the caller.

MCCTPI is called by the I/O interrupt handler in MCCINI when an interrupt occurs on a TP line. The LNEBLOK for the line which genrerated the interrupt is located. If it defines a BSC line, MCCBSC is called to process the interrupt. Upon return, MCCTPI returns to MCCINI.

For asynchronous lines, MCCTPI functions much like the Message-Reciever described in Part II. (Acknowledgements are placed on the front of SENDLIST so that they will be the next message sent on that line by MCCSND rather than being directly written to the line by MCCTPI. All conversion from the transparent message to MSGBLOK format is done by a call to routine MCCFLC.)

MCCBSC is responsible for all I/O activity on BSC lines. It gets control from MCCTPI when an interrupt is determined to be from a BSC line.

The CSW (in registers 2 and 3) is examined to see if any I/O errors occurred. If a bad channel status is present, the error is logged on the console and the failing operation retired. If the CSW indicates device errors, a SENSE CCW is

---

[28] This routine was supplied by the KSU Computing Center (where it was called FMGETSPL). There exists no source (with MCCS/370) for this routine. Any questions concerning it should be directed to the Systems Programming staff at the Computing Center.

executed on the line and control returns back to MCCTPI. When the interrupt indicating the SENSE has completed is received, one of the following actions is taken:[29]

1) Unit Exception: If the last operation was not a WRITE, a NAK is sent in response. If it was, a "fake" READ (the SKIP bit set in the CCW) is issued to clear the line of the incoming data which caused the bad status and control returns to MCCTPI. When the interrupt for the READ completes, the WRITE is reissued.

2) Unit Check: If the unit check was caused by a READ timeout (three seconds without receiving any data), a timecut counter in the LNEBLOK is incremented by 1. If sixteen consecutive timeouts have occurred, a message is logged on the console and a NAK resonse written. The timeout counter is zeroed and control passes to 4.

3) All other error conditions cause the CSW, SENSE data, and last CCW to be logged. Control then falls through to 4.

4) The last I/O operation is now determined. If the CSW last CCW address is zero, channel control check set, or the last CCW specifies a WRITE operation was being attempted, the current output buffer (in the LNEBLOK) is written to the line. READs causing a timeout are reissued (except for 16 consecutive timeouts as noted above). Any other error condition cause a NAK to be returned.

When the CSW indicates normal completion of the last I/O operation, a check is made to see if the completed operation was an ENABLE CCW. If it was, the LNEBLOK is initialized and a message displayed on the console indicating the line has signed on. A buffer is allocated (if available) to hold the first message that is received and its address placed in the LNEBLOK. If the line is acting in REMOTE mode a ACK0 (or fake message with the "W" bit on if no buffer was allocated above) is written to indicate to the host that MCCS/370 is ready for communication. For both HOST and REMOTE lines, a READ is issued to accept the first response.

Since anytime a message, ACK0, or NAK is written to the line, the WRITE CCW is chained to a READ CCW, normal status

---

[29] See the section entitled "Synchronous Data Adapter - Type II" in [IBM03] for the exact causes for each type of device error.

indicated by the CSW indicates a READ has completed (except for an ENABLE as noted above).

The contents of the input buffer (in the LNEBLOK) are then examined to find out what has been read. If the first character of the buffer is a NAK the contents of the output buffer are rewritten. If the buffer begins with DLE,STX a message has been received and is processed as described below. If the buffer contains ACK0 (DLE,X'70') control is passed to label BSCDEQ. Any other buffer contents cause a NAK tc be returned.

When the input buffer contains a message the FLAG2 field (Figure 5) is copied to the LNEBLOK field LNESTAT. If the "fake" bit is not on and the message is not a duplicate of the last message received, the address of the previously acquired buffer is obtained from the LNEELOK and the MSGBLOK built from the input buffer contents and placed on SENDLIST or RECVLIST as appropriate. POSTFLG is set to 1 and the wait bit in the I/O old PSW is turned off in case MCCSCH is waiting fcr something to do. Control then drops through to label ESCDEQ.

When ccntrol reaches label BSCDEQ, an ACK0 or message has just been received, either of which is a postive acknowledgement for the last thing sent. Therefore, if the output buffer contains a message (instead of ACK0), the message has been correctly received by another MCCS so it can be removed from SENDLIST using the MSGBLOK address contained in the LNEBLOK. An ACK0 is then placed in the output buffer in case no more messages are available for transmission.

Next, MCCBSC must determine what to respond with. If the "W" bit (Figure 6) in LNESTAT is on, control transfers to label BSCWAIT to initiate a two second timeout. The same is true if the line is held (set by the MCCS HOLD command). Otherwise SENDLIST is searched for MSBLOKs containing messages to be transmitted on this line. If one is found, it is written (the WRITE chained to a READ) and control returns to MCCTPI.

When a two second timeout is to be initiated (at label BSCWAIT), a call is made to entry-point MCCTIMS in routine MCCTIM. MCCTIMS builds a Timer Request Block (TRQBLOK) from the parameters passed in by MCCBSC. These parameters are the time interval (two seconds), a four byte "parameter" (the LNEBLOK address), and the address to which control is to be passed when the interval has expired (label MCCSBCW). This TRQBLOK is then sorted with existed PRQBLOKS according to the interval specified. The interval timer is set to the smallest interval so an external interrupt will occur in that amount of time causing MCCTIM to be entered by MCCINI's

external interrupt handler. MCCTIMS then returns to MCCBSC and it returns to MCCTPI. When the external interrupt occurs, MCCTIM will call the routine specified in the TRQBLOK with the "parameter" in register one. (In order to force a premature termination of the interval, the entry-point MCCTIMI in MCCTIM may be called. The value passed in through register 1 is compared with the "parameter" field of all the TRQBLOKs. If a match is found, that IRQBLOK is processed just as if its time interval had expired. This feature is used by MCCSND to inform MCCBSC that a MSGBLOK is available on SENDLIST for transmission with MCCBSC is waiting on the two second timeout.)

When MCCBSC is entered at MCCBSCW from MCCTIM, the LNEBLOK (address in register 1 from MCCTIM) is checked. If LNESTAT indicates the the "W" bit is off and that the line is not held, SENDLIST is again searched for a message to send. If one is found, it replaces the ACK0 in the output buffer, preceded by DLE,STX and followed by DLE,ETX(with a command chained WRITE, see [IBM03]).

If no buffer to hold the next message is allocated, an attempt is made to allocate one at this time. If it can't be allocated, the "W" bit in the FLAG2 field of the message is set. (A "fake" messsage may need to be created to set this bit if the output buffer contains ACK0). This bit indicates to another MCCS that MCCS/370 cannot accept a message in return.

Then the contents of the output buffer are written (either ACK0, a message, or a "fake" message) with the WRITE CCW chained to a READ CCW to accept the response. Control then returns to MCCTPI.

MCCCAI is entered from the I/O interrupt handler in MCCINI when an interrupt occurs on one of the CTCAs. If the CSW indicates ATTN status, a READ is issued to accept the data written to the CTCA by MCCCWR (executing in another virtual machine) when it coupled to MCCS/370s CTCA to pass along a SEND or RECV request for processing. MCCCAI then returns control to MCCINI.

When device end status is returned from the READ, an attempt is made to allocate a buffer to hold the PRQBLOK which was just read. If no buffers are available, MCCTIMS is called (as described for MCCBSC above) to request that MCCCAI be reentered (at label CAIRETRY) after two seconds so another attempt can be made to allocate the buffer. If the buffer is not available after the timeout, another timeout is requested. This continues until such time that a buffer is made available. Since the CTCA has not yet been decoupled from the virtual machine issuing the SEND or RECV, no other virtual machine can couple to MCCS's CTCA so MCCCAI need not

worry about being entered to process a different request on this CTCA. (Two other CTCAs exist for use by MCCDSP to indicate to a process that its request has be acted upon, however.)

After a buffer is allocated, the CTCA is decoupled so another virtual machine may utilize it. The PRQBLOK that was just read from the CTCA is placed in the buffer and chained to the end of WAITLIST. POSTFLG is set to "1" and the wait bit in the I/O old PSW turned off so that MCCSCH will call MCCDSP and MCCSND to process the newly acquired PRQBLCK. Control then returns to MCCINI.

MCCCON is entered from MCCINI's I/C interrupt handler on an interrupt from the virtual console. If the CSW indicates ATTN status, a READ is issued to allow the operator to enter a command. If the interrupt was caused by device end status being presented, the command buffer is scanned (for the command just read if a READ just completed or for additional commands, seperated by the newline character, if a write has just completed.) If the buffer is empty, control returns to MCCINI. Otherwise the command is processed and then control returns to MCCINI. Refer to the following section for a description of the commands accepted by MCCCON.

Operator Commands.

Several commands are available to control the execution of MCCS/370. These commands are entered by striking the ATTN key (or its equivalent) on the operator's console. A timestamp will be displayed and the keyboard unlocked (on 2741 terminals) so the command may be entered. Commands are free-format and operands are separated by one or more blanks. Multiple commands may be entered on the same physical line by using the current CP LINEND character [IBM07] This section describes the commands that are accepted and acted upon by routine MCCCON, the console handler. The minimum acceptable truncation for command names and operands is shown in upper case. All references to "line-id" specify the two character identifier of the line (and consequently the computer the line provides a link to) that the command is to effect.

Begin

This command causes the wait bit in the I/O old PSW to be turned off (if on). BEGIN is used to resume execution of MCCSCH when it is performing MCCS/370 initialization and has

halted to allow operator commands to be executed. It may also be used to cause MCCSCH to call MCCDSP and MCCSND if it is currently in a wait state.

CP command-line

The CP command specifed by "command-line" is passed to CP for execution using DIAGNOSE X'0008'. [IBM12] If the return code from CP is non-zero it will be displayed, otherwise the only response is that is that directly caused by execution of the command by CP.

Display Line line-id

This command causes information about the line specified by "line-id" to be displayed at the console. The line-id and device address, whether it is asynchronous or bisynchronous, and a descriptive term for most of the flags contained in the LNEBLCK are shown.

Display LINES

This command causes the line-id and device address for all the lines specified in the LNEBLOKs to be displayed. Detailed information about each individual line can be obtained by using the "DISPLAY LINE line-id" command.

Display Checkpt

The current setting of the checkpoint flag in MCCSECT (either "ON" or "OFF") when this command is issued.

Display Mccsid

This command causes the current two-character identifier for the computer MCCS/370 is running in to be displayed at the console.

Display Adcons

This command is used to display the names and entry-point address of all the MCCS modules.

DRain line-id

This command sets the drain flag in the LNEBLOK for the specified line to cause the disconnect sequence to begin.[30]

ENAble line-id

The command causes a DISABLE, SETMODE (BSC only), and ENABLE channnel program to be initiated to the line provided it is not already enabled. The ENABLE command is useful for restarting a line that has been DRAINed or HALTed or a BSC line that has just been attached to the MCCS virtual machine.

HALT line-id

This commands causes the specified line to be immediately disabled, whether it is active or not.

HOld line-id

This command prevents any further messages from being transmitted on the specifed line. Incoming messages are still accepted however. For BSC lines that are held, ACK0 and NAK will be the only data written on the line.

ICCate line-id

This command causes the memory address of the LNEBLOK for the line specified by line-id to be displayed. The CP DISPLAY command may then be issued, using the displayed address, to further interrogate the LNEBLOK contents.

RELease line-id

This command is used to counteract the effects of a previous HOLD command issued for the line. Any messages to be sent on the line are now eligible for processing by MCCSND and MCCBSC. (The MCCS BEGIN command should be issued following the RELEASE command to insure that MCCDSP and MCCSND will be

---

[30] MCCS/370 currently does not support the disconnect sequence. The DRAIN command is accepted but it is not acted upon by MCCSND, MCCTPI, or MCCBSC other than no messages will be sent on a line that is drained.

promptly called by MCCSCH.)

SET Checkpt {ON|OFF}

This command is used to control whether MCCCKP actually writes out the checkpoint data when it is called or just returns. Setting checkpoint OFF will result in a considerable performance improvement in MCCS, however a system failure will cause the contents of WAITLIST, SENDLIST, and RECVLIST to be lost. Checkpointing, by default, is ON.

SET Id line-id-1 line-id-2

This command is used to change the current line-id ("line-id-1") to another two character id ("line-id-2"). The line must not be active when this command is issued.

SET {Host|Remote} line-id

This command may be used to change the mode that a disabled BSC line is to operate in when it is subsequently enabled. (See the discussion of HOST and REMOTE BSC lines in Part II for more information.)

SET Mccsid new-id

This command is used to change the two character id of the computer (in relation to MCCS) that MCCS is running in. There must be no PRQBLOKs on WAITLIST or MSGBLOKs on SENDLIST and RECVLIST when this command is issued. (This is because the MCCSID is used to determine whether the TO_ID of a message indicates it is for a process on this computer or not, and therefore, whether the resultant MSGBLOK is placed on SENDLIST or RECVLIST.)

Testing Aids.

A routine named MCCMSG has been written to aid in the testing and debugging of MCCS/370. This routine executes in a CMS virtual machine and utilizes MCCSUM, MCCRUM, and MCCCWR to communicate with the MCCS virtual machine. MCCMSG is an interactive program which allows a "message"[31] or a file to be sent or received using MCCS/370.

MCCMSG will prompt the user to enter the information needed

to control its execution ("END" or "QUIT" cause MCCMSG to terminate and return to CMS.) so little additional information is needed here. A discussion of the format of the msg and file that MCCMSG sends and expects to receive is appropriate at this time however. If a new version of MCCS is implemented and the equivalent of MCCMSG is also written, then an easy means of testing the new system is avaliable by utilizing the "MCCMSG" in an existing MCCS implementation.

The format of the message buffer specified in the call to SEND for a msg is shown in Figure 8. The first 6 charcters identify the message as being a msg. Starting with the 7th character are 80 bytes (padded with blanks if necessary) of text. When a RECV is issued and this first six bytes of the returned MESSAGE parameter contain ':MSG ', the msg is simply displayed.

Sending and receiving files is considerably more complex. The first message that is sent contains a file header and is shown in Figure 9. The FILENAME and FILETYPE serve to identify the file being transmitted. They may need to be reformatted to conform to the file naming conventions of the receiving computer. (MCCMSG will prompt the user to enter a new file name if the one received is invalid or is identical to an existing CMS file.) Files of fixed or variable format are distinguished by a "F" or "V" in the F field of the header. The RLEN field is a four byte integer that specifies the length of the longest record in the file (in case a buffer needs to be allocated to hold the record as it is being reconstructed, etc.).

The file is then sent on the message(s) that follow the header. It is sent a continuous stream of characters, breaking the stream up into 128 bytes blocks so they may be placed in the MESSAGE parameter in the call to SEND. Each logical record in the file is preceded by a four byte count field that specifies the length of the record that follows (excluding the count field length). The last record of the file is followed by a count field contain zero to indicate end-of-file has be enccuntered.

MCCS/370 may be started in multiple virtual machines if desired. This feature is useful, for example, in testing changes using the modified version of MCCS/370 to communicate with an existing version running in another virtual machine which is known to work correctly. Using the MCCS SER command, the MCCSID and line-ids may be dynamically modified to "simulate" MCCS/370 communicating with a MCCS

---

[31] To avoid confusion between a MCCMSG "message" and a MCCS "message", the former will be spelled "msg".

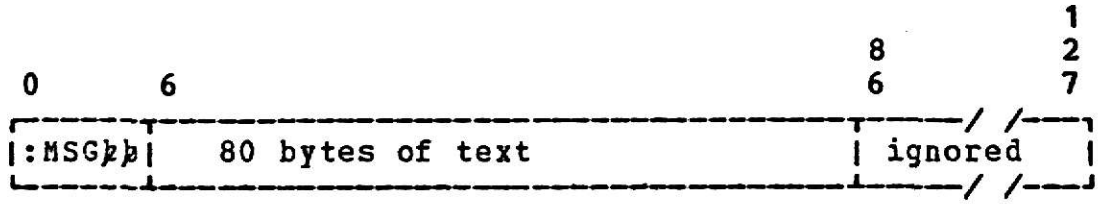


Figure 8: MCCMSG File Header Format

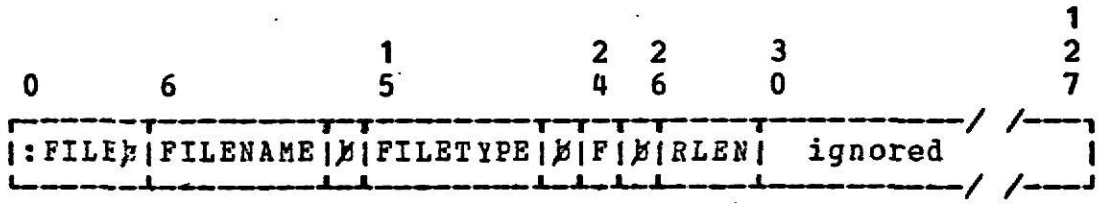


Figure 9: MCCMSG "msg" Header Format

that would normally execute in another computer.[32] If one dial-up BSC line and another dial-up or leased line are available, one may be attached to each MCCS virtual machine and the dial-up used to call the other line so that the BSC line protocol may be tested using only the 370.

System Correctness, Status, and Extensions.

The initial versions of MCCS/370 were tested by communication with an implementation of MCCS running on a Data General Nova mini-computer. This testing involved the use of asynchronous lines only.

Since this testing was done, support for BSC lines has been added and the message text size was expanded to 128 bytes. These modifications have not been tested using the NOVA MCCS at the time of this writing.

The original implementation of MCCS asynchronous protocol required an "available buffer count"[33] be returned as the

---

[32] MCCCWR must be modified to couple its CTCA to the correct virtual machine since the USERID in the COUPLE command assembled in MCCCWR normally specifies "VMJH8".

17th through the 32nd bits of the message text portion of the acknowledgement message. The 16th bit was then used as the flag designating this as a postive or negative acknowledgement. This bit has since been moved to the 1st bit of the message text and the buffer count done away with. (The code to support the buffer count still exists in MCCS/370 but the check to make sure it was positive before sending a message has been disabled. Now, when a message is received and no buffer space is available, a negative acknowledgement message is returned.)

The MCCS/370 BSC support has been tested by bringing up two MCCS virtual machines and having them communicate via to 2400 baud BSC lines, one line dialed into the other. Although this may not seem to be a ideal testing envircnment, since MCCS/370 was not aware that it was "talking to itself", it really should be sufficient to test the logic of the BSC routines. The only area of difficulty using this testing technique is that, even though the protocol implemented seems symmetrical, the protocol implemented indeed matches that described in Part II.

Any errors in MCCS/370 that were detected during the initial implementation and testing phase were corrected. It is unreasonable, however, to expect that all errors in a system of this size and complexity have been encountered and rectified. The asynchronous line support is the most likely candidate for "bugs" even though it has had the most rigorous testing. This is because since it was written, several major design changes have been made to MCCS. These changes caused considerable rearangement of code within MCCSND and MCCTPI and could have easily introduced some currently unencountered problems. (It is very likely that there is a problem with changes being made to SENDLIST while it is being processed by MCCSND thereby invalidating the absolute MSGBLOK pointer that it keeps. This is in the area where MCCSND enables interrupts temporarily between the processing of each MSGBLOK.)

MCCS/370 currently does not support the disconnect sequence for either asynchroncus or binary synchronous lines as described in Part II. (A flag in the LNEBLOK, "LNESTOP", was specifically defined for this purpose however.) The

---

[33] This count gave the number of buffers that were available for receiving messages when the response message was constructed. Then, before any message was transmitted, the buffer count of the MCCS that was to receive the message was checked. If it was positive, the message was sent and the count decremented; otherwise the message was held.

line may be shut down by disabling the line with the MCCS HALT command although it would be more desirable to modify MCCBSC and MCCTPI to properly support this feature.

No analysis has been made as to the performance of MCCS/370. Several casual observations have been made, however, and are noted in the following paragraphs.

Originally MCCS/370 was written to run as a user program in CMS. Interrupt handling became somewhat of a problem, both in being able to process the interrupt rapidly and in CMS's insistence on enabling interrupts at inopportune times (on entry to an external interrupt handler, for example, external interrupts are enabled!). A notable performance improvement (beyond the fact that interrupts could now be handled correctly) was noted when the switch to stand-alone operation was made. The areas of improvement were observed in CPU times, storage requirements, and working set size.

Taking a checkpoint each time a control block on WAITLIST, SENDLIST, or RECVLIST is changed causes a noticeable performance degradation (presumedly due to the I/O wait associated with spooling). The question of whether to run checkpointing or not has to be made by comparing the advantages of increased performance with the disadvantages of loss of one or messages when a system failure occurs. (This this can be disastrous when the message lost is in the middle of a data file that is being transmitted.)

BIBLIOGRAPHY

BIBLIOGRAPHY

[ABR75] Abrams, M.D., "Network Hardware Components", IEEE Computer Networks: Text and References for a tutorial, Chapter 3, New York, 1975, pp 3.1-3.9

[BAK??] Baker, Allen F., "A Deterministic Finite State Automaton for Binary Synchronous Communication", Bowling Green State University, Bowling Green, Ohio.

[BJO70] Bjorner, Dines, "Finite State Automation - Definitions of Data Communication Line Control Procedures", FJCC, 1970

[BLA75] Blanc, R.P., "Network Software Components", IEEE Computer Networks: Text and References for a Tutorial, Chapter 4, New York, 1975, pp 4.1-4.5

[CER74] Cerf, Vinton G., and Robert E. Kahn, A Protocol for Packet Network Intercommunication", IEEE Transactions on Communications, May 1974, pp 637-648

[DEN70] Dennis, Jack, "Modular Asynchronous Control Structures for a High Performance Processor", ACM Conference Record, 1970, pp 55-80

[DIJ65] Dijkstra, E. W., "Cooperating Sequential Processes", Technological University, The Netherlands, 1965

[ELO74] Elovitz, Honey S., and Constance L Heitmeyer, "What is a Computer Network?", National Telecommunications Conference 1974 Record, pp 1007-1014.

[IBM01] "General Information - Binary Synchronous Communication", IBM Corporation, Form A27-3004

[IBM02] "IBM System/370 Principles of Operation", IBM Corporation, Form A22-7000

[IBM03] "IBM 2701 Component Description", IBM Corporation, Form A22-6864

[IBM04] "IBM 2702 Component Description", IBM Corporation, Form A22-6846

[IBM05] "Introduction to the 3704 and 3705 Communications Controllers", IBM Corporation, Form A27-3051

[IBM06] "OS/VS2 HASP II Version 4 Logic", IBM Corporation, Form Y27-7255

[IBM07] "VM/370 Command Language For General Users", IBM Corporation, Form C20-1804

[IBM08] "VM/370 Control Program Logic", IBM Corporation, Form Y20-0880

[IBM09] "VM/370 Introduction", IBM Corporation, Form C20-1800

[IBM10] "VM/370 Planning and System Generation", IBM Corporation, Form C20-1801

[IBM11] "VM/370 RSCS Program Logic", IBM Corporation, Form Y20-0883

[IBM12] "VM/370 System Programmer's Guide", IBM Corporation, Form C20-1807

[IBM13] "VM/370 Terminal Users' Guide", IBM Corporation, Form C20-1810

[KSU75] "CP/CMS Guide", Kansas State University Computing Center, Manhattan Kansas, 1975

[MIN73] Mimo, N.W., et al, "Terminal Access to the ARPA Network: Experience and Improvements", Compton 73, pp 39-43

[SCH74] Schelonka, Edward P., Resource Sharing with ARPANET", National Telecommunications Conference 1974 Record, pp 1045-1048

[STU72] Stuzman, Byron, "Data Communications Control Procedures", Computing Surveys, Vol 4, NO 4, December, 1972

MCCS/370 Module Summaries

This and the following pages of this appendix give a brief description of each routine within MCCS.

Name:
MCCBSC

Function:
To handle all interrupts, reading, and writing for BSC IP lines.

Called by:
MCCTIM and MCCTPI

Entry points:
MCCBSCW

Entry conditions:
MCCBSC - R15 address of MCCBSC
R14 return address
R13 save area address
R4 interrupting device address
R2-R3 CSW from interrupt

MCCBSCW- R15 address of MCCBSCW
R14 return address
R13 save area address
R1 LNEBLOK address

Exit conditions:
None

Calls:
MCCCKP and MCCTIMS

External references:
MCCCTL and MCCPSA

Register usage:
R13 called routine save area address
R12 base
R11 base
R10 unused
R9 address of MCCSECT
R8 address of MSGBLOK
R7 address of LNEBLOK
R6-R5 unused
rest work

Name:
MCCCAI

Function:
Handle interrupts on the CTCAs, read PRQBLOKs from the CTCA, and add PRQELOKs to WAITLIST.

Called by:
MCCINI and MCCTIMS

Entry points:
CAIRETRY (not an external name)

Entry conditions:
MCCCAI - R14 return address
R12 address of MCCCAI
R4 interrupting device address
R2-R3 CSW at time of interrupt

CAIRETRY-R15 address of CAIRETRY
R14 return address
R13 save area address
R1 CTCA device address

Exit conditions:
None

Calls:
MCCCKP and MCCTIMS

External references:
MCCCTL and MCCPSA

Register usage:
R13 called routine save area address
R12 base
R11 base
R10 unused
R9 address of MCCSECT
R8 unused
R7 address of PRQBLOK
rest work

Name:
MCCCKP

Function:
To perform the checkpoint of WAITLIST, SENDLIST, and RECVLIST by writing their contents to the virtual punch.

Called by:
MCCBSC, MCCCAI, MCCDSP, and MCCSND

Entry points:
None

Entry Conditions:
R15 address of MCCCKP
R14 return address
R13 save area address
R1 'WAIT', 'SEND', or 'RECV'

Exit conditions:
None

Calls:
MCCGSI

External references:
MCCCTL and MCCPSA

Register usage:
R13 called routine save area address
R12 base
R11 base
R10 unused
R9 MCCSECT
R8-R4 unused
rest work

Name:
MCCCON

Function:
Handle interrupts from the virtual console and process MCCS/370 operator commands.

Called by:
MCCINI

Entry points:
None

Entry conditions:
R14 return address
R12 address of MCCCON
R4 virtual console address
R2-R3 CSW from interrupt

Exit conditions:
None

Calls:
None

External references:
MCCCIL and MCCPSA

Register usage:
R13 called routine save area address
R12 base
R11 base
R10 unused
R9 MCCSECT
R8 unused
R7 LNEBLOK
R6 unused
rest work

Name:
MCCCWR

Function:
Tc perform CTCA I/O and interrupt handling in virtual machines communicating with MCCS/370.

Entry points:
None

Entry conditions:
R15 address of MCCCWR
R14 return address
R13 save area address
R1 address of PRQBLOK to write

Exit conditions:
PRQBLOK has been updated per SEND/RECV request

Calls:
DMSKEY and HNDINT

External references:
NUCON

Register usage:
R13 called routine save area address
R12 base
R11 base
R10-R8 unused
R7 address of PRQBLOK
R6-R5 unused
rest work

Name:
MCCDSP

Function:
To examine PRQBLOKs, acting upon the SEND or RECV request, and, if necessary, sending a response to MCCCWR via the CTCA.

Called by:
MCCSCH

Entry points:
None

Entry conditions:
R15 address of MCCDSP
R14 return address

Exit conditions:
None

Calls:
MCCCKP

External references:
MCCCTL and MCCPSA

Register usage:
R13 called routine save area address
R12 base
R11 base
R10 unused
R9 MCCSECT
R8 address of MSGBLOK
R7 address of PRQBLOK
R4 unused
rest work

Name:
MCCEND

Function:
Tc mark the end of MCCS routines in memory. Contains no executable code.

Called by:
N/A

Entry points:
N/A

Entry conditions:
N/A

Exit cnditions:
N/A

Calls:
N/A

External references:
N/A

Register usage:
N/A

Name:
MCCFLC

Function:
To convert messages received from asynchronous TP lines to standard message format.

Called by:
MCCTPI

Entry points:
None

Entry conditions:
R15 address of MCCFLC
R14 return address
R13 save area address
R7 address of LNEBLOK
R1 pointer to message buffer
R0 length of message to convert

Exit conditions:
CC=0 if BCC count correct
CC=¬0 otherwise

Calls:
None

External references:
MCCCTL

Register usage:
R13 called routine save area address
R12 base
R11 base
R10 unused
R9 address of MCCSECT
R8 unused
R5 unused
rest work

Name:
MCCGSI

Function:
To return the spool id of the spool file active on the unit record output device specified by R0.

Called by:
MCCCKP

Entry points:
None

Entry conditions:
R15 address of MCCGSI
R14 return address
R0 address of unit record output device

Exit conditions:
R1 contains spool file id

Calls:
None

External references:
Various CP control blocks

Register usage:
R12 base
rest work

Name:
MCCINI

Function:
To perform the MCCS/370 virtual machine initialization.

Called by:
DMKLDOOE after reader IPL

Entry points:
MCCPSA, MCCCTL, and STARTUP

Entry conditions:
None

Exit conditions:
None

Calls:
MCCCAI, MCCCON, MCCSCH, MCCTIM, MCCTPI, SCVXXX, and SVC255

External references:
None

Register usage:
R15 base
R13 called routine save area address
R12 base
R11-R10 unused
R9 MCCSECT
R8-R5 unused
rest work

Name:
MCCRUM

Function:
To provide support for the MCCS RECV primitive for processes running in virtual machine executing CMS.

Called by:
user processes

Entry points:
RECV

Entry conditions:
R15 address of MCCRUM (RECV)
R14 return address
R13 save area address
R1 standard OS/360 parameter list

Exit conditions:
Parameter list updated as required.

Calls:
MCCCWR

External references:
None

Register usage:
R13 called routine save area address
R12 base
R11 base
R10-R8 unused
R7 address of PRQBLOK
R6-R3 unused
rest work

THIS BOOK WAS BOUND WITHOUT PAGE 68.

THIS IS AS RECEIVED FROM CUSTOMER.

Name:
MCCSND

Function:
To search SENDLIST for messages to be transmitted on asynchronous TP lines and to notify MCCBSC (through MCCTIMI) of a message ready to transmit on a BSC TP line.

Called by:
MCCSCH

Entry points:
None

Entry conditions:
R15 address of MCCSND
R14 return address
R13 save area address

Exit conditions:
None

Calls:
MCCCKP, MCCTLC, and MCCTIMI

External references:
MCCCTL and MCCPSA

Register usage:
R13 called routine save area address
R12 base
R11 base
R10 unused
R9 address of MCCSECT
R8 address of MSGBLOK
R7 address of LNEBLOK
R5 unused
R3-R2 unused
rest work

Name:
MCCSUM

Function:
To provide support for the MCCS SEND primitive for processes running in a virtual machine executing CMS.

Called by:
User processes

Entry points:
SEND

Entry conditions:
R15 address of MCCSUM (RECV)
R14 return addres
R13 save area address
R1 standard OS/360 parameter list

Exit conditions:
Parameter list updated as required.

Calls:
MCCCWR

External references:
None

Register usage:
R13 called routine save area address
R12 base
R11 base
R10-R8 unused
R7 address of PRQBLOK
R6-R3 unused
rest work

Name:
MCCTIM

Function:
To handle timer (interval and clock comparator) interrupts and provide delayed reentry to MCCS routines after specified interval or upon request.

Called by:
MCCBSC, MCCCAI, MCCINI, and MCCSND

Entry points:
MCCTIMI and MCCTIMS

Entry conditions:
MCCTIM - R12 address of MCCTIM
R14 return address

MCCTIMI- R15 address of MCCTIMI
R14 return address
R13 save area address
R1 TRQBLOK seach parameter value

MCCTIMS- R15 address of MCCTIMS
R14 return addres
R13 save area address
R2 address to pass control to at end of interval
R1 TRQBLOK parameter value
R0 time interval in hundredths of seconds

Exit conditions:
None

Calls:
None

External references:
MCCPSA

Register usage:
R13 unused
R12 base
R11 base
R10-R9 unused
R8 address of Timer Request Block (TRQBLOK)
R7-R4 unused
rest work

Name:
MCCTLC

Function:
To convert messages to the required format for transmission on asynchronous TP lines.

Called by:
MCCSND

Entry points:
None

Entry conditions:
R15 address of MCCTLC
R14 return address
R13 save area address
R1 pointer to buffer to convert
R0 length of data in buffer

Exit conditions:
R0 contains length of converted message

Calls:
None

External references:
MCCCTL

Register usage:
R13 called routine save area address
R12 base
R11 base
R10 unused
R9 address of MCCSECT
rest work

Name:
MCCTPI

Function:
Ic handle interrupts on asynchronous TP lines including the reading of incoming messages, adding the MSGBLOK to SENDLIST or RECVLIST, and placing the proper response MSGBLOK on SENDLIST.

Called by:
MCCINI

Entry points:
None

Entry conditions:
R14 return address
R12 address of MCCTPI

Exit conditions:
None

Calls:
MCCFLC

External references:
MCCCTL and MCCPSA

Register usage:
R13 called routine save area address
R12 base
R11 base
R10 unused
R9 address of MCCSECT
R8 address of MSGBLOK
R7 address of LNEBLOK
R6-R5 unused
rest work

MCCS ABEND CODES

The following list shows all the ABEND codes that can occur during MCCS/370 execution. When any of these conditions occur, a dump of virtual storage is taken using the CP DUMP command and MCCS is restarted. The dump title line will contain the ABEND code and the address at which the error occurred. By using this list, the proper routine may be found and by looking at that routine and the ABEND dump, the error can be analized. (The registers at the time of the error can be found at label GPRLOG in MCCPSA.)

Code: BSC001

Issued by: MCCBSC

Reason: A message was received but the LNEBLOK did not contain the address of a buffer in which to build the MSGBLOK.

Code: BSC002

Issued by: MCCBSC

Reason: The output buffer contained a message that was acknowledged but the LNEBLOK did not contain a valid MSGBLOK pointer.

Code: CAI001

Issued by: MCCCAI

Reason: A non-zero condition code was obtained when the SIO was issued to read the PRQBLOK from the CTCA.

Code: CAI002

Issued by: MCCCAI

Reason: An unexpected error status was received from an I/O operation on the CTCA.

Code: CKP001

Issued by: MCCCKP

Reason: Register 1 did not contain 'WAIT', 'SEND', or 'RECV' when MCCCKP was called to perform a checkpoint.

Code: CKP002

Issued by: MCCCKP

Reason: A non-zero condition code was obtained while attempting to write the checkpoint data to the virtual punch.

Code: DSP002

Issued by: MCCDSP

Reason: Sixteen attempts were made to write to the send response CTCA (X'120') with the SIO indicating the channel was busy.

Code: DSP003

Issued by: MCCDSP

Reason: Not operational status was returned from the SIO issued to the send response CTCA (X'120').

Code: DSP005

Issued by: MCCDSP

Reason: Sixteen attempts were made to write to the receive response CTCA (X'130') with the SIO condition code indicating the channel was busy.

Code: DSP006

Issued by: MCCDSP

Reason: Not operational status was returned from the SIO issued to the receive response CTCA (X'130').

Code: SCH001

Issued by: MCCSCH

Reason: A non-zero condition code was obtained from the SIO used to enable an asynchronous TP line.

Code: PRGxxx

Issued by: MCCSCH

Reason: A program check occurred during MCCS execution. The interrupt code replaces xxx.

Code: SVCxxx

Issued by: MCCSCH

Reason: A SVC with the invalid code xxx was issued.

Code: SND001

Issued by: MCCSND

Reason: Not operational status was returned from the HIO issued to a asynchronous line used to terminate the active INHIBIT.

Code: SND002

Issued by: MCCSND

Reason: A non-zero condition code was received from a SIO issued to a asynchronous line to write a message.

Code: SND100

Issued by:

Reason: No LNEBLOK could be found that provided the link specified by the TO_ID of a MSGBLOK on SENDLIST.

Code: IPI001

Issued by: MCCTPI

Reason: An acknowledgement was received for the last message transmitted but the LNEBLOK does not contain a valid MSGBLOK pointer.

VM/370 Dependencies

There are several features of VM/370 that MCCS/370 utilizes (beyond those provided in the simulation of a real 370 using the virtual machine concept). Some of these features may well change with a new Release or PLC of CP or CMS. The features felt most likely to be changed will be noted here so that they may be incorporated in MCCS/370 concurrently with the new version of CP or CMS.

MCCSCH used the CP SFBLOK DSECT [IBM08] and the spool file manipulation DIAGNOSE X'0014' [IBM12] to find the checkpoint spool files in the virtual card reader when MCCS is starting up. The SFBLOK macro in the MCCS MACLIB must be updated to reflect any changes made to the SFBLOK format.

MCCS/370 uses the VMFLOAD module and the CP IPLable loader. [IBM10] Both of the routines must be available and function similar to what they do in Release 2 PLC 13 for MCCS to be loaded.

The VM/370 update procedures (utilizing the CMS UPDATE command [IBM07] and the VMFASM EXEC [IBM10]) are used for applying updates to the MCCS/370 source. This facility must continue to be available to apply existing and future updates.

MCCS/370 also uses some modifications that have been made to CP and CMS by the Kansas State University Computing Center. One of these modifications, prefixing a CP command with a period to suppress most of the console output normally generated by that command, is used throughout MCCS.

KSU also has a modification to CP which causes the distribution code specified on the CP CLOSE command to be used (rather than ignored) when the printer or punch has been spooled to "*". This distribution code is used as a selection criteria by MCCSCH when it is checking for checkpoint spool files in the virtual reader. (The dependency on this modification may be removed by deleting the CLC and BNE statements in MCCSCH at sequence numbers 00890000 and 00900000 as the other checks that are made are sufficient to select the proper spool file.)

In order for MCCS/370 to restart, CMS is IPLed so that VMFLOAD may be used to create a new system load deck. A KSU modification to CMS allows a parameter of the form "EX xxxxx" to be specified in the PARM option of the IPL command and recognized by CMS when it is IPLed. CMS will automatically invoke the EXEC named "xxxxx" after it has

performed its initialization. (See the description of MCCSCH in Part III to see how this feature is used by MCCS.)

As mentioned in the description of the MCCCKP routine in Part III, the spool id of the spool file active on the punch is needed so that it may be purged when it is no longer needed. A routine named MCCGSI is called to return this elusive spool id. This routine was obtained from the KSU Computing Center (where it was named "FMGETSPL") and questions concerning it should be directed to the Systems Programming Staff there. (MCCGSI exists in TEXT deck only in MCCS/370, no source file is present.)

MCCS/370 Update Procedures

MCCS/370 is written in IBM System 370 assembler language and the source code exists as CMS files in the KSU File Manager System. The MCCS macro library is also a CMS file (named "MCCLIB MACLIB"). All the MCCS source and macro library are kept in COPYFILE [IBM07] "packed" format and therefore must be unpacked before re-assembly.

All changes made to MCCS should be done using the VM/370 source update procedures. [IBM10] (The MCCS control file is named "MCCR10 CNTRL", the AUX files "fn AUXMCC", and the updates are numbered sequentially starting with "fn MCCS0C01".)

There are many advantages to using this update technique over directly changing the source file by EDITing [IBM07] so it should be continued if at all possible. By using this procedure not only is a detailed log of all modifications automatically available, but modifications can easily be exchanged between different installations which run MCCS/370.

MCCS/370 Control Blocks

The fcllowing assembler lisings show the names and offsets of all the globally used control blocks in MCCS/370.

```
FILE: MCCBLOKS MCCR10   A1  KANSAS STATE UNIVERSITY - CMS V2 PLC 13 - 10/31/75 20:53          PAGE 001

TEXT MACS MCCLIB CMSLIB OSMACRO
TEXT AUXR10

*         MCCLIB   MACLIB   D1 SCRTCH 01/04/76  22:42
*         CMSLIB   MACLIB   S2 CMS190 02/25/75  09:04
*         OSMACRO  MACLIB   S2 CMS190 10/01/75  17:31
*         MCCBLOKS ASSEMBLE D1 SCRTCH 01/01/76  16:20
```

LOC OBJECT CODE ADDR1 ADDR2 STMT SOURCE STATEMENT ASM 0105 04.42 01/08/76

```
                                    2          MCCPSA                                                    00030000
000000                              3+MCCPSA   DSECT                                                    00040000
                                    5+*        MACHINE USAGE                                             00100000
000000                              7+IPLPSW   DS    D          INITIAL PROGRAM LOADING PSW            00120000
000008                              8+IPLCCW1  DS    D          INITIAL PROGRAM LOADING CCW1           00130000
000010                              9+IPLCCW2  DS    D          INITIAL PROGRAM LOADING CCW2           00140000
000018                    00000     10+        ORG   IPLPSW                                            00150000
000000                              11+RSTNPSW DS    D          PSW RESTART NEW PSW                    00160000
000008                              12+RSTOPSW DS    D          PSW RESTART OLD PSW                    00170000
000010                              13+NUCRSV0 DS    F          RESERVED FOR FUTURE USE                00180000
000014                              14+ASYSREF DS    A          ADDRESS OF SYSTEM ADDRESS TABLE        00190000
000018                              16+EXTOPSW DS    D          EXTERNAL OLD PSW                       00210000
000020                              17+SVCOPSW DS    D          SUPERVISOR CALL OLD PSW                00220000
000028                              18+PGMOPSW DS    D          PROGRAM OLD PSW                        00230000
000030                              19+MCKOPSW DS    D          MACHINE-CHECK OLD PSW                  00240000
000038                              20+IOOPSW  DS    D          I/O OLD PSW                            00250000
000040                              22+CSW     DS    D          CHANNEL STATUS WORD                    00270000
000048                              23+CAW     DS    F          CHANNEL ADDRESS WORD                   00280000
00004C                              25+NUCRSV1 DS    F          RESERVED FOR FUTURE USE                00300000
000050                              26+TIMER   DS    F          INTERVAL TIMER                         00310000
000054                              27+NUCRSV2 DS    F          RESERVED FOR FUTURE USE                00320000
000058                              29+EXTNPSW DS    D          EXTERNAL NEW PSW                       00340000
000060                              30+SVCNPSW DS    D          SUPERVISOR CALL NEW PSW                00350000
000068                              31+PGMNPSW DS    D          PROGRAM NEW PSW                        00360000
000070                              32+MCKNPSW DS    D          MACHINE-CHECK NEW PSW                  00370000
000078                              33+IONPSW  DS    D          I/O NEW PSW                            00380000
000080                              35+CPULOG  DS    48D        CPU LOGOUT AREA                        00400000
000200                    00080     36+        ORG   CPULOG                                            00410000
000080                              37+NUCRSV3 DS    2D         RESERVED FOR FUTURE USE                00420000
000090                              38+NUCRSV4 DS    F          RESERVED FOR FUTURE USE                00430000
000094                              39+MCNCLASS DS   H          N*1 - MONITOR CALL CLASS NUMBER        00440000
000096                              40+PERCODE DS    H          N*2 - PROGRAM EVENT RECORDER CODE      00450000
000098                              41+PERADDR DS    F          PROGRAM EVENT RECORDER ADDRESS         00460000
00009C                              42+MONCODE DS    F          MONITOR CALL CODE                      00470000
0000A0                              43+NUCRSV5 DS    4D         RESERVED FOR FUTURE USE                00480000
0000C0                              45+LOWSAVE DS    XL160      SAVE AREA FOR 1ST 160 BYTES OF STORAGE 00500000
000160                              47+FPRLOG  DS    4D         FLOATING POINT REGISTER LOGOUT AREA    00520000
000180                              48+GPRLOG  DS    16F        GENERAL PURPOSE REGISTER LOGOUT AREA   00530000
0001C0                              49+ECRLOG  DS    16F        EXTENDED CONTROL REGISTER LOGOUT AREA  00540000
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  ASM 0105 04.42 01/08/76
                                    51+*        MCCS USAGE                                                    00560000
000200                              53+EXTSAVE  DS    16F          SAVE AREA FOR EXTERNAL INTERRUPT HANDLER   00580000
000240                              54+SVCSAVE  DS    16F          SAVE AREA FOR SVC HANDLER                  00590000
000280                              55+IOSAVE   DS    16F          SAVE AREA FOR I/O INTERRUPT HANDLER        00600000

0002C0 00000000                     57+FREELOW  DC    A(0)         TOP FREE BLOCK ON FREE STORAGE STACK       00620000
0002C4 00000000                     58+FREEHI   DC    A(*-*)       HIGHEST VIRTUAL STORAGE ADDRESS            00630000
0002C8 00000000                     59+FREEHWM  DC    V(MCCEND)    HIGHEST STORAGE ADDRESS ALLOCATED          00640000
0002CC 000000A8                     60+FREESIZE DC    A(168)       SIZE OF STORAGE BLOCKS TO ALLOCATE         00650000

0002D0                              62+         DS    0D                                                        00670000
0002D0 C8C87AD4D47AE2E2             63+SYSTEMID DC    C'HH:MM:SS MM/DD/YY  MCCS VERSION 1.0 - 01/08/76'       00680000
```

```
                                    65          HCCSECT                                                    00040000
                                    66+HCCSECT  DSECT                                                      00090000
000000 C200000020C0C09C             67+CHANRCCW CCW   READ,*-*,SLI,L'CSBUFF    CCW TO READ FROM CTCA     00110000
000008 C100C00020C0CC00             68+CHANWCCW CCW   WRITE,*-*,SLI,*-*       CCW TO WRITE TO CTCA      00120000

000010                              70+WAITLIST DS    A             POINTER TO FIRST PRQBLOK ON WAITLIST    00140000
000014                              71+WAITEND  DS    A             POINTER TO LAST PRQBLOK ON WAITLIST     00150000
000018                              72+SENDLIST DS    A             POINTER TO FIRST MSGBLOK ON SENDLIST    00160000
00001C                              73+SENDEND  DS    A             POINTER TO LAST MSGBLOK ON SENDLIST     00170000
000020                              74+RECVLIST DS    A             POINTER TO FIRST MSGBLOK ON RECVLIST    00180000
000024                              75+RECVEND  DS    A             POINTER TO LAST MSGBLOK ON RECVLIST     00190000

000028                              77+MSGCOUNT DS    F             COUNTER USED TO GENERATE UNIQUE MSG IDS 00210000

00002C                              79+LASTWAIT DS    H             SPOOL ID OF LAST WAITLIST CHECKPT FILE  00230000
00002E                              80+LASTSEND DS    H             SPOOL ID OF LAST SENDLIST CHECKPT FILE  00240000
000030                              81+LASTRECV DS    H             SPOOL ID OF LAST RECVLIST CHECKPT FILE  00250000

000032                              83+CSBUFF   DS    CL156         BUFFER FOR SEND CTCA (X'100' AND X'120') 00270000
0000CE                              84+CRBUFF   DS    CL156         BUFFER FOR RECV CTCA (X'110' AND X'130') 00280000

00016A                              86+LASTSCOP DS    C             LAST CCW OPCODE USED ON SEND CTCA       00300000
00016B                              87+LASTRCOP DS    C             LAST CCW OPCODE USED ON RECV CTCA       00310000
00016C                              88+POSTFLG  DS    X             FLAG SET TO 1 TO INDICATE HCCSND AND    00320000
                                    89+*                            HCCDSP SHOULD BE CALLED               00330000
00016D E8                           90+CKPFLG   DC    AL1(YES)      DEFAULT CHECKPOINTING ON                00340000
00016E E5D4                         91+OURID    DC    C'VM'         MESSAGES WITH THIS TOID ARE FOR US      00350000

000170 C020                         93+LOLINE   DC    Y(X'020')     LOWEST TP LINE ADDRESS                  00370000
000172 C024                         94+HILINE   DC    Y(X'024')     HIGHEST TP LINE ADDRESS                 00380000

000174                              96+CONSOLE  DS    Y(*-*)        VIRTUAL CONSOLE ADDRESS                 00400000

000176 1070                         98+DLEACK0  DC    0XL2'00',AL1(DLE,ACK0)  POSITIVE ACKNOWLEDGEMENT FOR BSC 00420000
000178 1002                         99+DLESTX   DC    0XL2'00',AL1(DLE,STX)   TRANSPARENT START OF BSC TEXT   00430000
00017A 1003                        100+DLEETX   DC    0XL2'00',AL1(DLE,ETX)   TRANSPARENT END OF BSC TEXT     00440000
00017C 32321070                    101+ACKODATA DC    0XL4'00',AL1(SYN,SYN,DLE,ACK0)    DATA FOR POSITIVE ACK 00450000
000180 32323DF0                    102+NAKDATA  DC    0XL4'00',AL1(SYN,SYN,NAK,PAD)     DATA FOR NEGATIVE ACK 00460000
000184 40                          103+ITBMODE  DC    X'40'         SET 3705 IN ITB MODE                    00470000
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT  SOURCE STATEMENT
000185 C00000
000188 2F0CC0006CC0CC01            105+ENAB020  CCW   DISABLE,0,SLI+CC,1  DISABLE THE LINE                          00490000
000190 230001846C0CCC01            106+         CCW   SETMODE,ITBMODE,SLI+CC,L'ITBMODE  SET BSC ITB MODE          00500000
000198 270000002C0CCC01            107+         CCW   ENABLE,0,SLI,1 THEN ENABLE THE LINE                          00510000
0001A0 0A0002C82CC0012C            108+INHIB020 CCW   INHIBIT,BUFF020,SLI,L'BUFF020 INHIBIT FOR TELE LINES       00520000
0001A8 C10002C820C0CC00            109+WRITE020 CCW   WRITE,BUFF020,SLI,*-*     WRITE BUFFER FOR TELE LINES      00530000
0001B0 020002C820C0012C            110+READ020  CCW   READ,BUFF020,SLI,L'BUFF020     READ RESPONSE FOR TELE LIN  00540000

0001B8 C10003346C0CCC96            112+BWRN020  CCW   WRITE,BUFFA020,SLI+CC,L'DLESTX+MSGDATA                      00560000
0001C0 0100017A6C0CCC02            113+         CCW   WRITE,DLEETX,SLI+CC,L'DLEETX                               00570000
0001C8 C2000208C2C0012C            114+         CCW   READ,BUFF020,SLI,L'BUFF020                                 00580000
0001D0 C10003346000CC04            115+BWRA020  CCW   WRITE,BUFFA020,SLI+CC,L'ACKODATA                           00590000
0001D8 0200020B2C000I2C            116+         CCW   READ,BUFF020,SLI,L'BUFF020                                 00600000
0001E0 C10001806C0CCC04            117+BWRN020  CCW   WRITE,NAKDATA,SLI+CC,L'NAKDATA                             00610000
0001E8 020002C82C000I2C            118+         CCW   READ,BUFF020,SLI,L'BUFF020                                 00620000
0001F0 040003E22000CC01            119+BSEB020  CCW   SENSE,SEN020,SLI,L'SEN020                                  00630000

0001F8                             121+CSW020   DS    D                                                          00650000
000200                             122+CCW020   DS    D                                                          00660000

000208                             124+BUFF020  DS    CL300                                                      00680000
000334                             125+BUFFA020 DS    CL150                                                      00690000

0003CC                             127+BUFA020  DS    A                                                          00710000
0003D0                             128+MSGA020  DS    A                                                          00720000

0003D4 E5D6                        130+ID020    DC    C'NO'          THIS IS THE LINK TO THE NOVA                00740000
0003D6 8000                        131+LMID020  DC    XL2'8000'                                                  00750000
0003D8 4040                        132+LFID020  DC    C'  '                                                      00760000
0003DA C001                        133+BCNT020  DC    H'1'                                                       00770000
0003DC C000                        134+TCC020   DC    H'0'                                                       00780000

0003DE 81                          136+FLG1020  DC    AL1(LNEBSC+LNEHOST)   THIS IS A HOST BISYNC LINE            00800000
0003DF                             137+FLG2020  DS    X                                                          00810000
0003E0                             138+STAT020  DS    X                                                          00820000
0003E1                             139+LOP020   DS    X                                                          00830000
0003E2                             140+SEN020   DS    X                                                          00840000
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT
0003E3 C00CC0C0C0
0003E8 2F0CC0006C0CCC01          142+ENABO21  CCW   DISABLE,0,SLI+CC,1  DISABLE THE LINE                          00860000
0003F0 230001846CC0CCC01         143+         CCW   SETMODE,ITBMODE,SLI+CC,L'ITBMODE  SET BSC ITB MODE          00870000
0003F8 270C0000200CCCC01         144+         CCW   ENABLE,0,SLI,1 THEN ENABLE THE LINE                          00880000
000400 CA000468200C0012C         145+INHIBO21 CCW   INHIBIT,BUFFO21,SLI,L'BUFFO21 INHIBIT FOR TELE LINES       00890000
000408 C100046820C0CC00          146+WRITEO21 CCW   WRITE,BUFFO21,SLI,*-*     WRITE BUFFER FOR TELE LINES      00900000
000410 020004682000012C          147+READO21  CCW   READ,BUFFO21,SLI,L'BUFFO21     READ RESPONSE FOR TELE LIN 00910000

000418 C1000594600CCC96          149+BWBMO21  CCW   WRITE,BUFFAO21,SLI+CC,L'DLESTX+MSGDATA                      00930000
000420 C100017A60C0CC02          150+         CCW   WRITE,DLEETX,SLI+CC,L'DLEETX                                 00940000
000428 020004682000012C          151+         CCW   READ,BUFFO21,SLI,L'BUFFO21                                   00950000
000430 C1000594600C0CC04         152+BWBAO21  CCW   WRITE,BUFFAO21,SLI+CC,L'ACKODATA                             00960000
000438 020004682CC0012C          153+         CCW   READ,BUFFO21,SLI,L'BUFFO21                                   00970000
000440 C1000180600C0CC04         154+BWBNO21  CCW   WRITE,NAKDATA,SLI+CC,L'NAKDATA                               00980000
000448 020004682000C12C          155+         CCW   READ,BUFFO21,SLI,L'BUFFO21                                   00990000
000450 C4000642200C0CC01         156+BSENDO21 CCW   SENSE,SENO21,SLI,L'SENO21                                     01000000

000458                           158+CSWO21   DS    D                                                            01020000
000460                           159+CCWO21   DS    D                                                            01030000

C00468                           161+BUFFO21  DS    CL300                                                        01050000
C00594                           162+BUFFAO21 DS    CL150                                                        01060000

C00628                           164+BUFAO21  DS    A                                                            01080000
000630                           165+MSGAO21  DS    A                                                            01090000

C00634 C9C4                      167+IDO21    DC    C'ID'          THIS IS THE LINK TO THE INTERDATA             01110000
000636 8000                      168+LNIDO21  DC    XL2'8000'                                                    01120000
C00638 4040                      169+LFIDO21  DC    C'  '                                                        01130000
C0063A C001                      170+BCNTO21  DC    H'1'                                                         01140000
00063C C000                      171+TCCO21   DC    H'0'                                                         01150000

00063E 81                        173+FLG1O21  DC    AL1(LNEBSC+LNEHOST)   THIS IS A HOST BISYNC LINE           01170000
00063F                           174+FLG2O21  DS    X                                                            01180000
000640                           175+STATO21  DS    X                                                            01190000
C00641                           176+LCPO21   DS    X                                                            01200000
C00642                           177+SENO21   DS    X                                                            01210000
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT
000643 C00CCCC000
000648 2F0CC0006CCCCC01             179+ENAB022  CCW   DISABLE,0,SLI+CC,1  DISABLE THE LINE                        01230000
000650 230001846CC0CC01             180+         CCW   SETMCDE,ITBMCDE,SLI+CC,L'ITBMODE  SET BSC ITB MODE       01240000
000658 270CC00020C0CC01             181+         CCW   ENABLE,0,SLI,1 THEN ENABLE THE LINE                        01250000
000660 CA0006C82C00012C             182+INHIB022 CCW   INHIBIT,BUFF022,SLI,L'BUFF022 INHIBIT FOR TELE LINES      01260000
000668 C10006C82CC0CC00             183+WRITE022 CCW   WRITE,BUFF022,SLI,*-*     WRITE BUFFER FOR TELE LINES      01270000
000670 C20006C82CC0012C             184+READ022  CCW   READ,BUFF022,SLI,L'BUFF022    READ RESPONSE FOR TELE LIN 01280000

000678 C10007F460C0CC96             186+BWRM022  CCW   WRITE,BUFFA022,SLI+CC,L'DLESTX+MSGDATA                    01300000
000680 0100017A6CC0CC02             187+         CCW   WRITE,DLEETX,SLI+CC,L'DLEETX                             01310000
000688 C2000BC82000012C             188+         CCW   READ,BUFF022,SLI,L'BUFF022                               01320000
000690 C10007F46CC0CC04             189+BWRA022  CCW   WRITE,BUFFA022,SLI+CC,L'ACKODATA                         01330000
000698 C2000BC82CC0012C             190+         CCW   READ,BUFF022,SLI,L'BUFF022                               01340000
0006A0 C10001806CC0CC04             191+BWRN022  CCW   WRITE,NAKDATA,SLI+CC,L'NAKDATA                          01350000
0006A8 C2000BC82CC0012C             192+         CCW   READ,BUFF022,SLI,L'BUFF022                               01360000
0006B0 C40008A220C0CC01             193+BSEB022  CCW   SENSE,SEN022,SLI,L'SEN022                               01370000


0006B8                              195+CSW022   DS    D                                                        01390000
0006C0                              196+CCW022   DS    D                                                        01400000

0006C8                              198+BUFF022  DS    CL300                                                    01420000
0007F4                              199+BUFFA022 DS    CL150                                                    01430000

00088C                              201+BUFA022  DS    A                                                        01450000
000890                              202+MSGA022  DS    A                                                        01460000

000894 E7E7                         204+ID022    DC    C'XX'          THIS LINE IS UNASSIGNED                   01480000
000896 8000                         205+LMID022  DC    XL2'8000'                                                01490000
000898 4040                         206+LFID022  DC    C'  '                                                    01500000
00089A C001                         207+BCNT022  DC    H'1'                                                     01510000
00089C C000                         208+TCC022   DC    H'0'                                                     01520000

00089E 80                           210+FLG1022  DC    AL1(LNEBSC)    THIS IS A BISYNC LINE                     01540000
00089F                              211+FLG2022  DS    X                                                        01550000
0008A0                              212+STAT022  DS    X                                                        01560000
0008A1                              213+LCP022   DS    X                                                        01570000
0008A2                              214+SEN022   DS    X                                                        01580000
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0105 04.42 01/08/76
0008A3 C000C00000
0008A8 2F0C00006CC0CC01              216+ENAB023  CCW   DISABLE,0,SLI+CC,1  DISABLE THE LINE                    01600000
0008B0 C3000000h0C0CC01              217+         CCW   NOP,0,SLI+CC,1       NOP FOR ALIGNMENT WITH BSC ENABLE   01610000
0008B9 270CC00020CCCC01              218+         CCW   ENABLE,0,SLI,1 THEN ENABLE THE LINE                      01620000
0008C0 CA00C9282000012C              219+INHIB023 CCW   INHIBIT,BUFF023,SLI,L'BUFF023 INHIBIT FOR TELE LINES    01630000
0008C8 0100092H2CC0CC00              220+WRITE023 CCW   WRITE,BUFF023,SLI,*-*     WRITE BUFFER FOR TELE LINES  01640000
0008D0 0200C9282CC0012C              221+READ023  CCW   READ,BUFF023,SLI,L'BUFF023     READ RESPONSE FOR TELE LIN 01650000

0008D8 01000A5460C0CC96              223+BWRM023  CCW   WRITE,BUFFA023,SLI+CC,L'DLESTX+MSGDATA                   01670000
0008E0 C100017A60C0CC02              224+         CCW   WRITE,DLEETX,SLI+CC,L'DLEETX                            01680C00
0008E8 0200C9282CC0012C              225+         CCW   READ,BUFF023,SLI,L'BUFF023                              01690000
0008F0 01000A5460C0CC04              226+BWRA023  CCW   WRITE,BUFFA023,SLI+CC,L'ACKODATA                        01700000
0008F8 0200092B2CC0012C              227+         CCW   READ,BUFF023,SLI,L'BUFF023                              01710000
000900 C100018A6CC0CC04              228+BWRN023  CCW   WRITE,NAKDATA,SLI+CC,L'NAKDATA                          01720000
000908 0200092B2000012C              229+         CCW   READ,BUFF023,SLI,L'BUFF023                              01730000
000910 04000B0220C0C001              230+BSEB023  CCW   SENSE,SEN023,SLI,L'SEN023                               01740000

000918                               232+CSW023   DS    D                                                       01760000
000920                               233+CCW023   DS    D                                                       01770000

C00928                               235+BUFF023  DS    CL300                                                   01790000
C00A54                               236+BUFFA023 DS    CL150                                                   01800000

C00AEC                               238+BUFA023  DS    A                                                       01820000
C00AF0                               239+MSGA023  DS    A                                                       01830000

C00AF4 E3F1                          241+ID023    DC    C'T1'          THIS IS TELETYPE LINE 1                  01850000
C00AF6 8000                          242+LMID023  DC    XL2'8000'                                               01860000
C00AF8 4040                          243+LFID023  DC    C'  '                                                   01870000
C00AFA C001                          244+BCNT023  DC    H'1'                                                    01880000
C00AFC C000                          245+TCC023   DC    H'0'                                                    01890000

C00AFE 00                            247+FLG1023  DC    AL1(0)         THIS IS A TELETYPE LINE                  01910000
C00AFF                               248+FLG2023  DS    X                                                       01920000
C00B00                               249+STAT023  DS    X                                                       01930000
C00B01                               250+LOP023   DS    X                                                       01940000
C00B02                               251+SEN023   DS    X                                                       01950000
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                        ASM 0105 04.42 01/08/76
000B03 C000000000
000B08 2F0000006C0CCC01                  253+ENAB024  CCW   DISABLE,0,SLI+CC,1  DISABLE THE LINE                    01970000
000B10 C30CC0C06000CC01                  254+         CCW   NOP,0,SLI+CC,1      NOP FOR ALIGNMENT WITH BSC ENABLE  01980000
000B18 270CC00020C0CC01                  255+         CCW   ENABLE,0,SLI,1 THEN ENABLE THE LINE                     01990000
000B20 0A000B882CC0012C                  256+INHIB024 CCW   INHIBIT,BUFF024,SLI,L'BUFF024 INHIBIT FOR TELE LINES  02000000
000B28 C1000BH82CC0CC00                  257+WRITE024 CCW   WRITE,BUFF024,SLI,*-*    WRITE BUFFER FOR TELE LINES  02010000
000B30 02000B882CC0012C                  258+READ024  CCW   READ,BUFF024,SLI,L'BUFF024    READ RESPONSE FOR TELE LIN 02020000

000B38 01000CB46C0CCC96                  260+BWRN024  CCW   WRITE,BUFFA024,SLI+CC,L'DLESTX+MSGDATA                  02040000
C00B40 0100017A6CC0CC02                  261+         CCW   WRITE,DLEETX,SLI+CC,L'DLEETX                           02050000
000B48 C2000B882CC0012C                  262+         CCW   READ,BUFF024,SLI,L'BUFF024                             02060000
000B50 C1000CB46CC0CC04                  263+BWRA024  CCW   WRITE,BUFFA024,SLI+CC,L'ACKODATA                       02070000
000B58 02000B882CC0012C                  264+         CCW   READ,BUFF024,SLI,L'BUFF024                             02080000
000B60 C10001B06000CC04                  265+BWRN024  CCW   WRITE,NAKDATA,SLI+CC,L'NAKDATA                        02090000
000B68 0200UB882CC0012C                  266+         CCW   READ,BUFF024,SLI,L'BUFF024                             02100000
000B70 04000D622CC0CC01                  267+BSEB024  CCW   SENSE,SEN024,SLI,L'SEN024                             02110000


000B78                                   269+CSW024   DS    D                                                      02130000
000B80                                   270+CCW024   DS    D                                                      02140000

000B88                                   272+BUFF024  DS    CL300                                                  02160000
000CB4                                   273+BUFFA024 DS    CL150                                                  02170000

000D4C                                   275+BUFA024  DS    A                                                      02190000
000D50                                   276+MSGA024  DS    A                                                      02200000

000D54 E3F2                              278+ID024    DC    C'T2'          THIS IS TELETYPE LINE 2                 02220000
000D56 8000                              279+LMID024  DC    XL2'8000'                                              02230000
000D58 4040                              280+LFID024  DC    C'  '                                                  02240000
000D5A C001                              281+BCNT024  DC    H'1'                                                   02250000
000D5C C000                              282+TCC024   DC    H'0'                                                   02260000

000D5E C0                                284+FLG1024  DC    AL1(0)         THIS IS A TELETYPE LINE                 02280000
000D5F                                   285+FLG2024  DS    X                                                      02290000
000D60                                   286+STAT024  DS    X                                                      02300000
000D61                                   287+LCP024   DS    X                                                      02310000
000D62                                   288+SEN024   DS    X                                                      02320000
```

LOC OBJECT CODE ADDR1 ADDR2 STMT SOURCE STATEMENT ASM 0105 04.42 01/08/76

```
                        00027   290+ENABLE   EQU   X'27'                    02340000
                        0002F   291+DISABLE  EQU   X'2F'                    02350000
                        0000A   292+INHIBIT  EQU   X'0A'                    02360000
                        00002   293+READ     EQU   X'02'                    02370000
                        00001   294+WRITE    EQU   X'01'                    02380000
                        00004   295+SENSE    EQU   X'04'                    02390000
                        00008   296+TIC      EQU   X'08'                    02400000
                        00023   297+SETMODE  EQU   X'23'                    02410000
                        00003   298+NOP      EQU   X'03'                    02420000

                        00020   300+SLI      EQU   X'20'                    02440000
                        00010   301+SKIP     EQU   X'10'                    02450000
                        00040   302+CC       EQU   X'40'                    02460000
                        00008   303+CE       EQU   X'08'                    02470000
                        00004   304+DE       EQU   X'04'                    02480000
                        00002   305+UC       EQU   X'02'                    02490000
                        00001   306+UE       EQU   X'01'                    02500000
                        00080   307+ATTN     EQU   X'80'                    02510000
                        00004   308+CCC      EQU   X'04'                    02520000
                        00001   309+TIMEOUT  EQU   X'01'                    02530000


                        000E8   311+YES      EQU   C'Y'                     02550000
                        000D5   312+NO       EQU   C'N'                     02560000
                        000D5   313+NONE     EQU   C'N'                     02570000


                        000B1   315+CR       EQU   X'B1'                    02590000
                        00070   316+ACKO     EQU   X'70'                    02600000
                        00010   317+DLE      EQU   X'10'                    02610000
                        00003   318+ETX      EQU   X'03'                    02620000
                        0003D   319+NAK      EQU   X'3D'                    02630000
                        000F0   320+PAD      EQU   X'F0'                    02640000
                        00002   321+STX      EQU   X'02'                    02650000
                        00032   322+SYN      EQU   X'32'                    02660000
                        00000   323+R0       EQU   0                        02670000
                        00001   324+R1       EQU   1                        02680000
                        00002   325+R2       EQU   2                        02690000
                        00003   326+R3       EQU   3                        02700000
                        00004   327+R4       EQU   4                        02710000
                        00005   328+R5       EQU   5                        02720000
                        00006   329+R6       EQU   6                        02730000
                        00007   330+R7       EQU   7                        02740000
                        00008   331+R8       EQU   8                        02750000
                        00009   332+R9       EQU   9                        02760000
                        0000A   333+R10      EQU   10                       02770000
                        0000B   334+R11      EQU   11                       02780000
                        0000C   335+R12      EQU   12                       02790000
                        0000D   336+R13      EQU   13                       02800000
                        0000E   337+R14      EQU   14                       02810000
                        0000F   338+R15      EQU   15                       02620000
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT.  SOURCE STATEMENT                                    ASM 0105 04.42 01/08/76
                                    340          LNEBLOK                                                          00050000
000000                              341+LNEBLOK  DSECT                                                            00030000
000000 2F0000006CC0CC01             342+LNENAB   CCW   DISABLE,0,SLI+CC,1  DISABLE FOR TELE/BSC LINES            00040000
000008 2300018460C0CC01             343+         CCW   SETMODE,ITBMODE,SLI+CC,L'ITBMODE (NOP FOR TELE LINES)    00050000
000010 2700C0C020C0CC01             344+         CCW   ENABLE,0,SLI,1 ENABLE FOR BSC AND TELE                   00060000

000018 0A000080200012C              346+INHIBCCW CCW   INHIBIT,BUFF,SLI,L'BUFF  NO-TIMEOUT READ FOR TELE        00080000
000020 010CC08060C0CC00             347+WRITECCW CCW   WRITE,BUFF,SLI+CC,*-*     WRITE MESSAGE FOR TELE         00090000
000028 0200008020C0012C             348+READCCW  CCW   READ,BUFF,SLI,L'BUFF      READ WITH TIMEOUT FOR TELE     00100000

000030 010001AC6000CC96             350+LNEWRMSG CCW   WRITE,LNEOBUF-LNEBLOK,SLI+CC,L'DLESTX+MSGDATA BSC WRITE  00120000
000038 0100017A60C0CC02             351+         CCW   WRITE,DLEETX-MCCSECT,SLI+CC,L'DLEETX  LEAVE BSC TRANSP.  00130000
000040 0200008020CCCC96             352+LNERDMSG CCW   READ,LNEIBUF-LNEBLCK,SLI,L'LNEIBUF  READ BSC MSG/RESPONS 00140000
000048 C10001AC6CC0CC04             353+LNEWRACK CCW   WRITE,LNEOBUF-LNEBLOK,SLI+CC,L'ACKODATA   WRITE BSC ACKO 00150000
000050 0200008020CCCC96             354+         CCW   READ,LNEIBUF-LNEBLCK,SLI,L'LNEIBUF  READ BSC MSG/RESPONS 00160000
000058 0100018060C0CC04             355+LNEWRNAK CCW   WRITE,NAKDATA,SLI+CC,L'NAKDATA      WRITE BSC NAK        00170000
000060 C200008020C0CC96             356+         CCW   READ,LNEIBUF-LNEBLOK,SLI,L'LNEIBUF  READ BSC MSG/RESPONS 00180000

000068 0400025A20C0CC01             358+LNESENC  CCW   SENSE,LNESENSE,SLI,L'LNESENSE BSC AND TELE SENSE         00200000

000070                              360+LNECSW   DS    D               LAST CSW FOR THIS LINE                   00220000
000078 C000000000000C00             361+LNECCWSA CCW   *-*,*-*,*-*,*-*      SAVE AREA FOR LAST CCW ON ERROR     00230000

000080                              363+BUFF     DS    CL300           TELE INPUT/OUTPUT BUFFER                 00250000
0001AC                    00080     364+         ORG   BUFF                                                     00260000
000080                              365+LNEIBUF  DS    CL150           BSC INPUT BUFFER                         00270000
000116                    001AC     366+         ORG                                                            00280000

0001AC                              368+ACKBUFF  DS    CL150           TELE BUFFER TO HOLD ACK/NAK               00300000
000242                    001AC     369+         ORG   ACKBUFF                                                  00310000
0001AC                              370+LNEOBUF  DS    CL150           BSC OUTPUT BUFFER                        00320000
000242                    00242     371+         ORG                                                            00330000


000244                              373+LNEMSGA  DS    A               ADDRESS OF MSGBLOK FOR LAST MESSAGE SENT 00350000
000248                              374+LNEBUFA  DS    A               ADDRESS OF NEXT BSC MSGBLOK BUFFER       00360000

00024C                              376+LNEID    DS    CL2             LINE ID OF THIS LINE                     00380000
00024E                              377+LNELMSID DS    H               MSGMSID OF LAST MESSAGE RECIEVED         00390000
000250                              378+LNELFRID DS    CL2             FIRST 2 CHARS OF MSGFRID OF LAST MESSAGE 00400000
000252                              379+LNEDCNT  DS    H               TELE BUFFER COUNT FOR OTHER MCCS         00410000
000254                              380+LNETOC   DS    H               BSC TIMEOUT COUNTER                      00420000

000256                              382+LNEFLG1  DS    X               FLAG BYTES FOR LINE STATUS               00440000

                          00080     384+LNEBSC   EQU   X'80'               THIS IS A BSC LINE                   00460000
                          00040     385+LNEON    EQU   X'40'               THIS LINE HAS SIGNED ON              00470000
                          00020     386+LNEACTV  EQU   X'20'               READ/WRITE ACTIVE ON TELE LINE       00480000
                          00010     387+LNERESP  EQU   X'10'               EXPECTING RESPONSE FOR TELE MSG      00490000
                          00008     388+LNEHOLD  EQU   X'08'               HOLD (DON'T SEND) MSGS ON THIS LINE  00500000
                          00004     389+LNEDRN   EQU   X'04'               THIS LINE IS DRAINED (DRAINING)      00510000
                          00002     390+LNEWTO   EQU   X'02'               BSC LINE WAITING FOR ACKO TIMEOUT    00520000
                          00001     391+LNEHOST  EQU   X'01'               THIS LINE IS IN HOST MODE            00530000
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT  SOURCE STATEMENT                                ASM 0105 04.42 01/08/76
000257                           393*LNEFLG2  DS    X          FLAG BYTE TWO                          00550000
                          00080  395*LNESTART EQU   X'80'           LINE WAITING FOR FIRST ACK/MSG   00570000
000258                           397*LNESTAT  DS    X          COMMUNICATION BYTE FOR OTHER HCCS (BSC) 00590000
                                 399**                         (BITS DEFINED IN MSGFLG2)              00610000
000259                           401*LASTOP   DS    X          LAST I/O OPERATION PERFORMED ON THIS LINE 00630000
00025A                           402*LNESENSE DS    X          SENSE BYTE FOR I/O ERRORS              00640000

000260                           404*         DS    0D                                                00660000
                          00260  405*LNESIZE  EQU   *-LNEBLOK   SIZE OF A LNEBLOK                     00670000
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                        ASM 0105 04.42 01/08/76
                                    407          MSGBLOK                                                          00060000
000000                              408+MSGBLOK  DSECT                                                           00030000
000000 C10C000C2000CC94             409+MSGCCW   CCW    WRITE,MSGTOID,SLI,MSGDATA CCW TO CKP THE BLOCK            00040000
000008 C8000000                     410+MSGTIC   DC     AL1(TIC),AL3(*-*)   TIC (ALSO CHAIN PTR) TO NEXT MSGBLOK 00050000
00000C                              411+MSGTOID  DS     CL6             ID OF PROCESSOR MESSAGE IS FOR            00060000
000012                              412+MSGFRID  DS     CL6             ID OF PROCESSOR MESSAGE IS FROM           00070000
000018                              413+MSGMSID  DS     H               UNIQUE IDENTIFIER FOR THIS MESSAGE        00080000
00001A                              414+MSGUSID  DS     H               IDENTIFIER ASSIGNABLE BY SENDER           00090000
00001C                              415+MSGFLG1  DS     X               FLAG BYTE                                 00100000
                          00080     416+MSGNOANS EQU    X'80'           DON'T EXPECT RESPONSE FROM THIS MESSAGE   00110000
                          00040     417+MSGNOFRE EQU    X'40'           DON'T ATTEMPT TO FREE THIS MSGBLOK        00120000
00001D                              418+MSGFLG2  DS     X               FLAG BYTE FROM ORIGINATING MCCS           00130000
                          00080     419+MSGDMSG  EQU    X'80'           THIS IS A DUMMY MESSAGE                   00140000
                          00040     420+MSGWAIT  EQU    X'40'           WAIT BEFORE SENDING ANOTHER MESSAGE       00150000
                          00020     421+MSGDISC  EQU    X'20'           DISCONNECT THE LINE                       00160000
00001E                              422+MSGTEXT  DS     CL128           TEXT OF MESSAGE                           00170000
0000A0                              423+         DS     0D                                                        00180000
                          00C94     424+MSGDATA  EQU    *-MSGTOID       LENGTH OF MESSAGE DATA IN MSGBLOK         00190000
                          000A0     425+MSGSIZE  EQU    *-MSGBLOK       LENGTH OF MESSAGE BLOCK                   00200000
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT  SOURCE STATEMENT                                    ASM 0105 04.42 01/08/76
                                    427          PRQBLOK                                                      00070000
000000                              428+PRQBLOK  DSECT                                                        00030000
000000 0100000C20C0CC9C             429+PRQCCW   CCW    WRITE,PRQUSER,SLI,PRQDATA  CCW TO CHECKPOINT BLOCK    00040000
000008 C8000000                     430+PRQTIC   DC     AL1(TIC),AL3(*-*)   TIC (ALSO CHAIN PTR) TO NEXT PRQBLOK 00050000
00000C                              431+PRQUSER  DS     CL8            USERID OF REQUESTING PROCESS            00060000
000014                              432+PRQVADDR DS     CL3            VADDR OF REQUESTING PROCESS'S CTCA      00070000
000017                              433+PRQTOID  DS     CL6            ID OF PROCESSOR SENDING/RECEVING MSG    00080000
00001D                              434+PRQFRID  DS     CL6            ID OF PROCESSOR MESSAGE IS FOR/FROM     00090000
000023                              435+PRQFLG1  DS     X              FLAG BYTE                               00100000
                          00080     436+PRQSEND  EQU    X'80'          ...PROCESS IS SENDING A MESSAGE         00110000
                          00040     437+PRQRECV  EQU    X'40'          ...PROCESS IS RECEVING A MESSAGE        00120000
                          00020     438+PRQWAIT  EQU    X'20'          ...PROCESS WILL WAIT TO RECIEVE MESSAGE 00130000

000024                              440+PRQMSID  DS     H              UNIQUE IDENTIFIER FOR THIS MESSAGE      00150000
000026                              441+PRQUSID  DS     H              IDENTIFIER ASSIGNABLE BY USER           00160000
000028                              442+PRQTEXT  DS     CL128          TEXT OF MESSAGE                         00170000
0000A8                              443+         DS     0D                                                     00180000
                          0009C     444+PRQDATA  EQU    *-PRQUSER      LENGTH OF DATA IN PRQBLOK               00190000
                          000A8     445+PRQSIZE  EQU    *-PRQBLOK      LENGTH OF PRQBLOK                       00200000
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                    ASM 0105 08.42 01/08/76
                                    447          END                                      00080000
```

SYMBOL LEN VALUE DEFN REFERENCES ASM 0105 04.42 01/08/76

```
ACKBUFF  00150 C0C0C1AC 00368  C0369
ACKO     00001 0000C070 00316  00098 00101
ACKODATA 00004 C0C00170 C0101  00115 00152 00189 00226 00263 00353
ASYSREF  00004 C000CC14 00014
ATTN     00001 C0C0CC80 00307
BCNT020  00002 00C003DA 00133
BCNT021  00002 0000063A C0170
BCNT022  00002 CCC0CE9A 00207
BCNT023  00002 C000CAFA 00244
BCNT024  00002 C000CE5A 00281
BSEB020  00008 00C0C1F0 00119
BSEB021  00008 00C0C450 00156
BSEB022  00008 C0C00EB0 C0193
BSEB023  00008 C000C910 00230
BSEB024  00008 C0C0CE70 C0267
BUFA020  00004 CCC0C3CC 00127
BUFA021  00004 C0C00E2C C0164
BUFA022  00004 CCC0CE8C 00201
BUFA023  00004 CCC00AEC 00238
BUFA024  00004 C0C00E4C 00275
BUFF     00300 C000CC80 00363  00346 00346 00347 00348 00348 00364
BUFFA020 00150 C0C0C314 00125  00112 00115
BUFFA021 00150 C0C0C594 00162  00149 00152
BUFFA022 00150 0CC007E4 00199  00186 00189
BUFFA023 00150 C0CC0A54 00236  00223 00226
BUFFA024 00150 C0C00CE4 00273  00260 00263
BUFF020  00300 C0C00208 00124  00108 00108 00109 00110 00110 00114 00114 00116 00116 00118 00118
BUFF021  00300 C0C00468 00161  00145 00145 00146 00147 00147 00151 00151 00153 00153 00155 00155
BUFF022  00300 C00006C8 C0198  00182 00182 00183 00184 00184 00188 00188 00190 00190 00192 00192
BUFF023  00300 C0000928 00235  00219 00219 00220 00221 00221 00225 00225 00227 00227 00229 00229
BUFF024  00300 CCC0CEE8 C0272  00256 00256 00257 00258 00258 00262 00262 00264 00264 00266 00266
BWRA020  00008 CCC001D0 00115
BWRA021  00008 CC00C430 00152
BWRA022  00008 C0C00690 00189
BWRA023  00008 CC00CEF0 C0226
BWRA024  00008 00000E50 00263
BWRM020  00008 CCC001E8 C0112
BWRM021  00008 C0C0C418 C0149
BWRM022  00008 C0000678 C0186
BWRM023  00008 C000C6D8 00223
BWRM024  00008 C0C00E38 C0260
BWRN020  00008 00C001E0 00117
BWRN021  00008 CCC0C440 C0154
BWRN022  00008 CC0006A0 00191
BWRN023  00008 C0C0C5C0 00228
BWRN024  00008 C0000E60 00265
CAM      00004 C0C0CC48 C0023
CC       00001 CCC0CC40 00302  00105 00106 00112 00113 00115 00117 00142 00143 00149 00150 00152 00154 00179 00180 00186
                               00187 00189 00191 00216 00217 00223 00224 00226 00228 00253 00254 00260 00261 00263 00265
                               00342 00343 00347 00350 00351 00353 00355
CCC      00001 C000CCC4 00308
CCW020   00008 00000200 00122
CCW021   00008 C000C460 00159
CCW022   00008 C00006C0 00196
CCW023   00008 CC0C0520 00233
```

SYMBOL LEN VALUE DEFN REFERENCES ASM 0105 04.42 01/08/76

```
CCW024   00008 00C0CE80 00270
CE       00001 0000C008 00303
CHANRCCW 00008 00C0CC00 00067
CHANWCCW 00008 00C0CC08 00068
CKPFLG   00001 C0000160 00090
CONSOLE  00002 C0000174 00096
CPULOG   00008 00C0CC80 00035  00036
CR       00001 0000CCB1 00315
CRBUFF   00156 CCCCCCCE 00084
CSBUFF   00156 00C0CC32 00083  00067
CSW      00008 00C0CC40 00022
CSW020   00008 C00001F8 00121
CSW021   00008 00C00458 C0158
CSW022   00008 00C006B8 00195
CSW023   00008 C000C518 C0232
CSW024   00008 00000E78 00269
DE       00001 C000CC04 00304
DISABLE  00001 C000CC2F 00291  00105 00142 00179 00216 00253 00382
DLE      00001 00C0CC10 00317  00098 00099 00100 00101
DLEACK0  00002 00C00176 00098
DLEETX   00002 00C0017A C0100  00113 00113 00150 00150 00187 00187 00224 00224 00261 00261 00351 00351
DLESTX   00002 00C00178 C0099  00112 00149 00186 00223 00260 00350
ECHLOG   00004 00C001C0 00049
ENABLE   00001 0000CC27 00290  00107 00144 00181 00218 00255 00344
ENAB020  00008 C000C168 C0105
ENAB021  00008 00C003E8 00142
ENAB022  00008 C0000E48 C0179
ENAB023  00008 00C0C8A8 00216
ENAB024  00008 C0000E08 C0253
ETX      00001 00C0CC03 00318  00100
EXINPSW  00008 00C0CC58 C0029
EXTOPSW  00008 C0CCCC18 C0016
EXTSAVE  00004 C0C002C0 00053
FLG1020  00001 C00003DE 00136
FLG1021  00001 C0C0063E C0173
FLG1022  00001 0000C89E 00210
FLG1023  00001 CC000AFE C0247
FLG1024  00001 C0C00D5E 00284
FLG2020  00001 C0C003DF 00137
FLG2021  00001 C0C0063F 00174
FLG2022  00001 C0000E9F C0211
FLG2023  00001 00000AFF 00248
FLG2024  00001 C0C0CD5F C0285
FRRLOG   00008 C0000160 00047
FREEHI   00004 C0C002C4 00058
FREEHWM  00004 000002C8 00059
FREELOW  00004 C000C2C0 00057
FREESIZE 00004 000002CC 00060
GPRLOG   00004 C0C0C180 00048
HILINE   00002 C0000172 00094
ID020    00002 C0C003D4 00130
ID021    00002 C0000634 00167
ID022    00002 C000CE94 C0204
ID023    00002 00000AF4 00241
ID024    00002 C000CD54 C0278
```

ASM 0105 04.42 01/08/76

SYMBOL	LEN	VALUE	DEFN	REFERENCES
SEN022	00001	C0C00EA2	00214	00193 00193
SEN023	00001	C0000E02	00251	00230 00230
SEN024	00001	CCC00E62	00288	00267 00267
SETMODE	00001	C0C0CC23	00297	00106 00143 00180 00343
SKIP	00001	CCC0CC10	C0301	
SLI	00001	CCC0CC20	00300	00067 00068 00105 00106 00107 00108 00109 00110 00112 00113 00114 00115 00116 00117 00118 00119 00142 00143 00144 00145 00146 00147 00149 00150 00151 00152 00153 00154 00155 00156 C0179 00180 00181 00182 00183 00184 00186 00187 00188 00189 00190 00191 00192 00193 00216 00217 00218 00219 00220 00221 00223 00224 00225 00226 00227 00228 00229 00230 00253 00254 C0255 00256 00257 00258 00260 00261 00262 00263 00264 00265 00266 00267 00342 00343 00344 00346 00347 00348 00350 00351 00352 00353 00354 00355 00356 00358 00409 00429
STAT020	00001	C00003E0	00138	
STAT021	00001	C0000640	00175	
STAT022	00001	C0C00EA0	C0212	
STAT023	00001	CCC00E00	00249	
STAT024	00001	CCC00E60	00286	
SIX	00001	CCC0CC02	00321	00099
SVCNPSW	00008	00C0CC60	C0030	
SVCOPSW	00008	C0C0CC20	00017	
SVCSAVE	00004	C0000240	C0054	
SYN	00001	CCC0CC32	00322	00101 00101 00102 00102
SYSTEMID	00046	C0C0C2D0	C0063	
TIC	00001	C0C0CC0A	00296	00410 00430
TIMEOUT	00001	CCC0CCC1	C0309	
TIMER	00004	C000CC50	00026	
TCC020	00002	C00003DC	C0134	
TCC021	00002	C0C00638	00171	
TCC022	00002	C0C0CE9C	C0208	
TCC023	00002	C0000A1C	00245	
TCC024	00002	C0000E5C	C0282	
UC	00001	C0C0CC02	00305	
UE	00001	C0C0CC01	C0306	
WAITEND	00004	0000CC14	00071	
WAITLIST	00004	00C0CC10	C0070	
WRITE	00001	C0C0CC01	00294	00068 00109 00112 00113 00115 00117 00146 00149 00150 00152 00154 00183 00186 00187 00189 00191 00220 00223 00224 00226 00228 00257 00260 00261 00263 00265 00347 00350 00351 00353 00355 00409 00429
WRITECCW	00008	00C0CC20	00347	
WRITE02C	00008	C0C001A8	00109	
WRITE021	00008	C0C0C408	C0146	
WRITE022	00008	C0000E68	C0183	
WRITE023	00008	C0C0CEC8	C0220	
WRITE024	00008	00000E28	C0257	
YES	00001	C0C0CCE8	C0311	00090

ASM 0105 04.42 01/08/76

```
NO STATEMENTS FLAGGED IN THIS ASSEMBLY
HIGHEST SEVERITY WAS     0
OPTIONS FOR THIS ASSEMBLY
  ALIGN, ALOGIC, BUFSIZE(STD), NODECK, ESD, FLAG(0), LINECOUNT(55), LIST, NOMCALL
  NOMLOGIC, NUMBER, OBJECT, NORENT, RLD, STMT, NOLIBMAC, TERMINAL, NOTEST, XREF
  SYSPARM()
WORK FILE BUFFER SIZE = 7294
TOTAL RECORDS READ FROM SYSTEM INPUT          8
TOTAL RECORDS READ FROM SYSTEM LIBRARY      467
TOTAL RECORDS PUNCHED                         0
TOTAL RECORDS PRINTED                       829
```

MCCS

The Design and Implementation of a Multi-Computer Communications System

by

SHELDON LEE FOX

B.S., Kansas State University, 1973

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1976

ABSTRACT

This paper describes MCCS, a Multi-Computer Communications System. MCCS allows processes executing in one or more computers to communicate, via "messages", with processes executing in the same or other computers.

MCCS provides two process-level primitives, SEND and RECV, to facilitate this inter-process communication. By keeping the MCCS-process interface simple, this system should be easily implemented on a varity of computer systems, whether they be a "mini" or "maxi".

This paper is divided into two parts. The first part defines the general MCCS specifications. This includes such things as the exact process parameter definitions, transmitted message format, and line protocol for both asynchronous and binary synchronous transmission. To aid in presenting these specifications, sample data structures and functional modules are given.

The second part describes an actual implementation of MCCS on an IBM 370/158 computer. This part serves as complete documentation of MCCS/370, as it is called, and is intended for someone wishing to use, modify, or enhance it. The Appendices supplement this part by providing additional implementation dependent information.