MULTICS TECHNICAL BULLETIN

To: Distribution

From: Bill Silver

Date: November 15, 1973

Subject: Proposed Changes to the SYSERR and OPERATOR'S CONSOLE Software

This document describes the changes that have been made to the syserr mechanism and the operator's console software. This MTB along with MOSN 4.3.1 Revision 1 and MOSN 11.3 obsoletes MSB 108. The issues discussed in this document are:

1. programs affected
2. the major changes
   a) new mechanism for logging syserr messages
   b) new syserr codes
   c) asynchronous use of the operator's console
   d) more convenient operator interface to the console
   e) recovery from an inoperative operator console
   f) improved BCD to ASCII to BCD transliteration and improved input canonicalization
   g) metering the usage of the wired log buffer and the two console write buffers
3. what still must be done.

THE PROGRAMS INVOLVED

The following programs have been changed:

1. syserr_real: This program has been completely rewritten. It no longer contains the code which interfaces with the operator's console. syserr_real is responsible for putting the syserr messages into the wired log buffer.
2. wired_hardcore_data: This wired ring 0 data base now contains all of the wired data needed to log syserr messages and to interface with the operator's console. This includes

a) various locks, counters, offsets, and meters

b) the dcw lists used to drive the operator's console

c) the wired log buffer

d) the operator's console read buffer and its two write buffers.

3. ocdim_: This program has been completely rewritten. It is the user ring operator's console dim. It has been programmed to use the operator's console as an asynchronous device.

4. privileged_mode_ut: A new entry has been added to this procedure: "smic_me". This entry will generate a specific interrupt on the processor on which it is running. It is called by syserr_real to generate the syserr log interrupt.

5. hphcs_: The following highly privileged hardcore gates have been added:

a) ocdcm_dim_read - called by ocdim_ to read a message from the operator's console.

b) ocdcm_dim_write - called by ocdim_ to write a message on the operator's console.

c) ocdcm_resetwrite - called by ocdim_ to delete all operator's console write messages queued by calls to hphcs_$ocdcm_dim_write.

d) ocdcm_dim_event_chan - called by ocdim_ to set up the event wait channel used to coordinate use of the operator's console.

e) ocdcm_err_event_chan - called by an answering service procedure to set up an event call channel that can be signalled when the operator's console becomes inoperative.

f) syserr_logger_init - called to turn off the init flag in the syserr_log partition.

The following highly privileged hardcore gates have been deleted:

a) syserr_store_event_channel

b) syserr_opr_con_write

c) syserr_opr_con_read

6. init_collections: This procedure was changed to call the two new initialization procedures: oc_data_init and syserr_log_init.

7. emergency_shutdown: This procedure was changed to reset certain data in wired_hardcore_data.

8. shutdown: This procedure was changed to reset the certain data in wired_hardcore_data.

9. scs: This data base was changed to include the new syserr log interrupt pattern.

10. scs_init: This procedure was changed to accept the new process interrupt defined on the INT configuration card. (See MOSN 4.3.1 Revision 1.)

11. sys_info: Several data fields used by the old syserr_real have been deleted.

The following new programs have been added:

1. syserr_log: A paged segment which resides in a special secondary storage partition - the PART LOG partition. (See MOSN 4.3.1 Revision 1.) This segment is where syserr messages are placed.

2. syserr_logger: This procedure handles the syserr log interrupt. It copies syserr messages from the wired syserr log into the paged segment syserr_log.

3. ocdcm_: This is the ring 0 dcm for the operator's console.

4. oc_data_init: This procedure initializes the data in wired_hardcore_data that is needed by ocdcm_ to interface with the operator's console.

5. syserr_log_init: This procedure initializes the data in wired_hardcore_data that is needed by syserr_real to log syserr messages. It also initializes the syserr_log partition.

6. meter_w_h_c_: This procedure is called to meter the usage of the three wired buffers in wired_hardcore_data: wired log buffer, syserr write buffer, olm write buffer.

7. oc_trans_output_: This procedure is called to convert an ASCII output message to a format that is suitable for writing on the operator's console.

8. oc_trans_input_: This procedure is called to convert input from the operator's console into a usable ASCII string.

The following programs have been deleted from the system:

1. read_convert

2. syserr_init

The following BOS include files have been changed. They are used by the BOS programs "loaddm", "setup" and "util":

1. readt.incl.alm: This program has been changed to lock the console keyboard after a 30 second timer interrupt.

2. error.incl.alm: This program has been changed to allow the request button to be used to unlock the console keyboard after it has been locked due to a 30 second timer interrupt.

THE MAJOR CHANGES

SYSERR LOGGING MECHANISM

The syserr messages provide a great deal of interesting information about the operation of the system -especially about error conditions which have occurred. There is currently no mechanism for keeping and analyzing these messages on-line.

The logging mechanism described here will try to save all syserr messages in a special log partition - the LOG partition. The following brief scenario shows how a syserr message is logged:

1. syserr_real is called to write the syserr message.

2. The message is in the form of an "ioa_" string with accompanying data arguments. The message is expanded by formline_.

3. The message is assigned a unique sequence number - in fact the next sequential sequence number.

4. syserr_real then tries to put the expanded ASCII message (the ASCII time field is not included) in the wired log buffer. This buffer contains a nonwrap around FIFO list of variable length entries. Each entry contains:

   a) the sequence number of this message

   b) its syserr code

c) the length of the expanded message text

d) the calendar clock time when the message was put into the wired log

e) the message text.

5. If there is room for this message entry in the wired log buffer, syserr_real will generate the syserr_log process interrupt.

6. syserr_real will then convert the message to output format (with the current console this is BCD) and will call ocdcm_ to output the message.

7. syserr_real and ocdcm_ are wired and masked from taking the syserr_log interrupt. When syserr_real returns and the syserr_log interrupt is unmasked, the interrupt will be taken and handled by syserr_logger.

8. syserr_logger will copy the whole wired log buffer into its stack. Then it will reset the wired log buffer. The next syserr message will be placed at the top of the wired log buffer.

9. syserr_logger will then copy the messages that were in the wired log buffer into the paged log partition.

The paged log partition is known as the segment "syserr_log". It can be accessed only in ring 0. The include file used to reference the syerr_log segment and any of its message entries is supplied as Table I.

When syserr_real tries to put a message into the wired log buffer, it may be full. The list of entries in the wired log buffer does not wrap around. Thus the message will not be put into the wired log buffer. This message will be lost from the log. In this case (unless the message has a special syserr code) the message will be written on the console with the following string prefixed before the time field of the message:

"*lost xxxxxx, z"

where xxxxxx is the sequence number of the message, and
z is its syserr code.

syserr_real still compares messages that are written with the

previous message written. If the messages are the same, the character "=" will be substituted for the text of the message. Note, the text that is placed in the log is also "=".

The log partition is mapped into the segment syserr_log each time the system is booted. However, the data in the partition is NOT reinitialized each time the system is booted. The messages logged can be saved across boots. The "init_flag" in the syserr_log controls whether or not the log will be reinitialized. If it is zero the syserr_log will be reinitialized when the system is booted. There are two ways to set this flag to zero:

1. In BOS type the TEST PART LOG WRITE command.

2. On-Line call the gate hphcs_$syserr_logger_init. This will cause the syserr_log to be reinitialized the next time the system is booted.

The following occurs when the data in the log partition is reinitialized:

1. The "init_flag" is set ON.

2. The sequence number is reset to zero. The next message logged by syserr_real will be assigned the sequence number: (1).

3. A dummy message entry is placed at the top of the syserr_log. It will be the first message in the log and will have a sequence number of zero.

The size of the wired log is coded into wired_hardcore_data.alm. It is currently set at 150 words. This is enough space to hold about ten average size message entries. To change the size of this buffer wired_hardcore_data.alm must be reassembled. No other procedure has to be changed.

NEW SYSERR CODES

In order to make full use of the logging mechanism several additional syserr codes have been defined. Below is a list of all the syserr codes:

CODE	ACTION
0	log, write message with no alarm, and return
1	log, write message with alarm, and crash system
2	log, write message with alarm, and

	terminate the process
3	log, write message with alarm, and return
4	log and return - do not write message unless the message has been lost from the log
5 - 9	log and return - do not write message even if it is lost from the log

Note, the action taken for a code 1 message has been changed slightly. Instead of crashing the system by calling pmut$bos, syserr_real will call pmut$bos_and_return. Thus, it is possible to reenter MULTICS from BOS via a GO. This implies that procedures which call syserr with a code of 1 may be returned to.

Note, code 4 messages which are written out on the console because they were lost from the log are not compared with the previous message written. The "=" mechanism applies only to messages with codes (0-3).

ASYNCHRONOUS USE OF THE OPERATOR'S CONSOLE

The most serious defect in the current operator's console software is that is can only use the console in a synchronous manner. When called to write a message (either a syserr message or a message from the Initializer process), if the console is already busy, the current syserr_real will loop waiting for the console to become free. It will not return until it has initiated the write operation. If it was called to read from the console it will also wait by looping until the console is free. It will not return until the read operation has been initiated. While syserr_real is looping waiting for the console it is using the exclusive services of one processor. On a one processor system the whole system will effectively be stopped. Some fixed size system queues may overflow. The system may crash. For these reasons the operator's console is seldom used by the operations staff while running a service system. However, syserr messages must be typed out on the operator's console. A burst of syserr messages can tie up the console - and thus the system - for up to a minute or more.

The new console software will almost completely eliminate the problems discussed above. By queuing write messages and by having the Initializer process go blocked when one of its requests cannot be serviced immediately, the system will almost never have to loop waiting for the operator's console to become free. The following brief scenarios describe how this is done.

Reading:

1. All read requests for the operator console must go through the "oc_" dim and thus ocdim_. It will call ocdcm_ to read one input line. No read ahead is performed.

2. Regardless of whether or not ocdcm_ can initiate a read operation at this time, it will return to ocdim_ telling it to go blocked.

3. ocdim_ will force the initializer process to go blocked.

4. When the read operation has completed, ocdcm_ will wakeup the initializer process.

5. ocdim_ will then call ocdcm_ again. The input line will be returned.

6. ocdim_ must then have the raw input data transliterated and canonicalized.

Writing a Dim Message:

1. The "oc_" dim (ocdim_) can be called to write a block of data containing several lines.

2. ocdim_ will process this data line by line. It will convert each line to output format.

3. It will then call ocdcm_ to write one line.

4. ocdcm_ will then try to put this line into a wired buffer that has been reserved just for dim write messages.

5. If there is room in the dim write buffer for this message, ocdcm_ will return, telling ocdim_ that the message has been queued. ocdim_ will then process the next line if there is one.

6. If this message could not be queued in the dim write buffer, ocdcm_ will return telling ocdim_ to block.

7. ocdim_ will force the initializer process to go blocked.

8. When a dim write message terminates and its space is freed up in the dim write buffer, ocdcm_ will wakeup the initializer process.

9. ocdim_ will then call ocdcm_ to write this same message.

Writing a Syserr Message

1. syserr_real calls ocdcm_ to write a message.

2. ocdcm_ will try to put this message into a wired buffer reserved just for syserr messages.

3. If there is room in the syserr write buffer for this message, ocdcm_ will return to syserr_real.

4. If the syserr message cannot be queued, ocdcm_ will NOT return to syserr_real. It must loop waiting for a syserr message to terminate writing and to free up enough space in the syserr write buffer for this message.

5. ocdcm_ cannot return to syserr_real until the message has been queued. Only when an unusually large burst of syserr messages have to be written will the system have to wait for the operator's console to become free.

ocdcm_ provides the coordination between reading and writing and between writing syserr messages and writing dim messages. Writing syserr messages is given priority over writing dim messages and writing either type of message is given priority over reading.

The two write buffers are contained in wired_hardcore_data as one single write buffer. At system initialization time the buffer is split into two parts to form the syserr write buffer and the dim write buffer. (See MOSN 4.3.1 Revision 1 for details of the OPC configuration card.) The read buffer is also contained in wired_hardcore_data. The size of these buffers can be changed only by reassembling wired_hardcore_data.alm. No other procedures have to be changed. The read buffer is large enough for one maximum size input line (14 words). The combined write buffer is 150 words. If it were split according to the default values (80% for the syserr write buffer), then the syserr write buffer would have enough room for about 12 or 13 average size syserr messages.

NEW OPERATOR INTERFACE

The new operator's console software makes it much easier to use the operator console. For details on this new interface see MOSN 11.3. Some of these features are more than just convenient. They make the operator's console safer to use

on service. The main feature involved is, of course, the fact that the operator's console is now programmed asynchronously. Two other features are also important the 30 second timer and the break mechanism.

A basic problem with the operator's console is that it cannot be used for writing while a read operation is in progress. Thus, no syserr messages can be written while the console keyboard is unlocked. With the current software, the operator must hit the End of Message button in order to terminate a read operation. If for any reason he fails to do this, the system will eventually crash. The 30 second timer feature solves this problem. If 30 seconds elapses without the operator typing a character or hitting the End of Message button, the timer will go off. The read operation will be terminated and the keyboard will be locked. It will then be available for writing.

The current software does not provide any mechanism for suppressing unwanted console output. If the operator types a command which produces a large amount of output, it may take several minutes for all of it to be typed. During this time, syserr messages could be written, but no other use can be made of the console. With the break mechanism described in MOSN 11.3 this is no longer a problem.

INOPERATIVE CONSOLE RECOVERY

With the current operator's console software, if the console becomes inoperative for any reason (power is off, repeated parity errors, etc.), no syserr messages can be written. No "oc _" dim calls can be performed.

It is a serious problem if the operator doesn't receive syserr messages. The new console software attempts to solve this problem. When ocdcm_ determines that the console is inoperative, it will try to signal the answering service via an event call channel. It will signal the sequence number of of the first syserr message that could not be written. The answering service procedure which receives this signal will try to do something about the problem. If there is no other terminal which it can use, then there is nothing that can be done. If there is another terminal available to the Initializer process, it may try to do the following:

1. Write a message telling the operator about the problem.

2. Write out all the syserr messages that could not be written by the operator's console. It will get these messages from the syserr_log partition.

3. Try to switch "oc_" dim to the replacement terminal.

ocdcm_ will signal the answering service each time it

is called to write a syserr message. If ocdcm_ determines that the console has become operative again it will signal the answering service to discontinue the emergency procedures.

Note, in order for this recovery mechanism to function the following conditions must be satisfied:

1. The answering service must have set up an event call channel which ocdcm_ can use to signal it.

2. The logging mechanism must be enabled since the answering service will get the syserr messages out of the syserr_log.

TRANSLITERATION AND CANONICALIZATION

A full description of the new transliteration and canonicalization features is provided in MOSN 11.3. Below are some notes about how these new features affect syserr messages:

1. Upper case letters are now expressed in a unique way. Thus pathnames and possibly other information found in syserr messages will be more meaningful.

2. Since output messages can be continued on a new line if they are longer than one line, syserr messages may be longer.

METERS

The procedure meter_w_h_$_ is used to meter the three wired buffers in wired_hardcore_data: wired log buffer, syserr write buffer, dim write buffer. The raw meter data for each of these buffers in also kept in wired_hardcore_data. The information which can be obtained from these meters is listed below:

1. total time metering performed

2. percentage of time the buffer was empty

3. percentage of time the buffer was full

4. time weighted average number entries in the buffer

5. total number of entries put into the buffer

6. maximum number of entries ever in the buffer at one time

7. average length of an entry

WHAT HAS TO BE DONE

In order to make full use of the facilities provided by the new syserr and console software the following additional facilities should be developed:

1. A mechanism to copy the syserr_log into a user ring and to sort, edit, and print the syserr messages.

2. A BOS program to copy into the log partition any messages left in the wired log buffer after the system crashed.

3. A procedure to copy the raw meter data out of ring zero and to compute and print the information available from this meter data.

4. Implement the answering service mechanism for handling an inoperative console.

5. Change BOS to put BOS error messages (disk errors, etc.) in the log.

6. Change the on-line Salvager to use the syserr logging mechanism.