

MULTICS STAFF BULLETIN- 108

TO: Distribution  
FROM: Bill Silver  
DATE: July 17, 1973  
SUBJECT: Proposed Changes to Syserr and the Operator Console Dims

This document describes the basic design of a new implementation of the operator console dim and the syserr mechanism. The basic goals which this new implementation is to achieve are:

1. Provide a facility for logging all syserr messages.
2. Provide a mechanism for queuing messages that are to be written on the operator console. In most cases this should eliminate the situation where the system is blocked waiting for the operator console to become free.
3. Provide a mechanism that results in the system never being blocked waiting for status while reading a message from the operator console.
4. Provide a new operational mode in which the request button can be used to signal a break and which allows the operator console to remain open and available for input as long as the operator is actively using the console.
5. Notify the operator that the operator console has become inoperable.
6. Establish a useful and consistent meaning for all characters typed on the operator console during both input and output.

The general scheme used to implement these new functions involves the following segments:

1. A ring 1 operator console dim: ocdim\_  
2. A ring 0 operator console dim: ocpcm\_

3. **Syserr\_real:** This procedure will no longer contain the code used to communicate with the operator console. It will contain all of the mechanism used to log the syserr messages.
4. **wired\_hardcore\_data:** a wired ring  $\emptyset$  data segment which contains:
  - a) various locks, counters, meters, etc.;
  - b) the dcw lists used to drive the operator console;
  - c) the buffers for operator console input and output messages;
  - d) the buffer where syserr messages are temporarily logged.
5. **syserr\_log:** a paged segment which resides in a special secondary storage partition. It is in this data segment where syserr messages are more permanently logged.
6. **oc\_trans\_:** replaces read\_convert\_ and the entry formline\_\$ge which will be deleted. This procedure transliterates ASCII strings into BCD strings and BCD strings into ASCII strings.
7. A new initializer procedure which is signalled when the operator console becomes inoperative. If possible it will notify the operator of this condition via some other terminal attached to the initializer process.
8. Several procedures which initialize or cleanup the syserr\_log or wired\_hardcore\_data.
9. The following procedures will be involved with the new syserr\_log but are not to be part of the initial implementation:
  - a) Ring 4 procedures which edit and print the logged syserr messages.
  - b) BOS procedures that will put output messages in the log.

### NEW FEATURES

#### 1. SYSERR LOG:

The syserr\_log segment is a ring  $\emptyset$  paged segment which resides in a special partition of secondary storage. This segment is used to hold a wraparound list of syserr messages. No message is permanently logged since eventually all messages will be overlaid. The partition should be large enough so that a message is not overlaid for several weeks.



## 2. OUTPUT MESSAGE BUFFERS:

The ring  $\emptyset$  dim "ocdcm\_" will now maintain two somewhat separate buffers for messages being written on the operator console. There will be one buffer for syserr messages "sys\_buf" and one buffer for ocdim\_ messages "dim\_buf". The size of these buffers is specified by the LOG configuration card. (See Table III)

When syserr\_real and ocdim\_ call ocdcm\_ to write a message they will call two different entries. Both entries must be passed the output message in BCD format. Since the size of the messages vary the number of messages that can be put in the write buffers will also vary.

When ocdcm\_ is called to write a syserr message it will try to find room for it in the syserr write buffer. Note, if there are no messages queued in the dim write buffer then the space used for this buffer will be added to the syserr write buffer. Until all syserr messages have been written no dim message can be queued.

If ocdcm\_ had room in the write buffer to put this syserr message then it will try to write the message. If the operator console happens to be busy ocdcm\_ will not wait for it to become free. It will just return to syserr\_real.

If there was no room for this message in the write buffers then ocdcm\_ will not return to syserr\_real. It will return only when the message can be put into the buffer. ocdcm\_ will loop waiting until a write operation has terminated and can be removed from the buffer thus making room for this message.

If ocdcm\_ was called by ocdim\_ to write a message it will look for room in only its write buffer. If there is room in the buffer it will try to write the message. It will return to ocdim\_ indicating that the message has been written.

If there is no room for this message in its write buffer then it will return indicating that the message was not written. In this case ocdim\_ will block itself. When ocdcm\_ determines that there is room for that message in the dim\_buf it will send a wake-up to ocdim\_ which will then call it back to write this same message.

Note, ocdcm\_ will receive all interrupts associated with the operator console.

Note, two special entries are now part of ocdcm\_:

- a) `ocdcm_$check`: is called by `syserr_real` to check on whether or not there are any `syserr` messages still queued. `ocdcm_` will go through the process of checking status to see if a write has terminated and if so taking that message out of the buffer and writing the next message.
- b) `ocdcm_$resetwrite`: is called by `ocdim_` when it is given a `resetwrite` command. All messages queued in the `dim_buf` -- except the one being written -- will be removed.

### 3. INPUT MESSAGES:

When `ocdim_` is called to read a message it will call `ocdcm_` to read the message. `ocdcm_` will then try to issue a read command to the console. Even if the console is busy and the read command cannot be initiated at that time `ocdcm_` will return. Thus at no time will the system loop waiting for the termination of the console read. In any case when `ocdcm_` returns to `ocdim_` it will indicate that no message has yet been read. `ocdim_` noting this will block itself.

When `ocdcm_` does complete the read operation it will send a wake-up to `ocdim_` which will then call back `ocdcm_` to get the input message.

### 4. NEW OPERATIONAL MODE:

A field in the LOG configuration card is used to specify the operational mode of the operator console. (See Table III.) There are now two ways in which the operator console can be used.

#### A. Service Mode

This mode involves using the operator console in exactly the same way that it has always been used. Its name implies that this will be the normal mode for a system that is running service. The important points about this mode are:

1. The operator console user must press the request button each time he wants to type an input message.
2. When the request button is pressed the console will not be unlocked for input until all write messages have been written. There is no way to suppress unwanted output.

## B. Development Mode

Development users may find this operator console mode useful. The use of the request button is changed.

1. The request button can be used to suppress unwanted initializer output. Note, syserr messages can never be suppressed. When the request button is pushed the current line being typed will be completed. The next line to be typed will also be completed. The remaining lines of output will be lost.
2. The user will not have to use the request button in order to type each line of input. Whenever all output messages have been written (even if the request button was not used) the console will be unlocked and available for input. The user can continue to type input lines and receive output without ever HAVING to use the request button. If the user has no more input he may leave input mode (lock the console) by typing the following line: "\$\*\$". This line will be recognized by ocpcm\_ as a quit. This line will not be passed back to ocdim\_. Once the console is locked the user will have to hit the request button in order to have it unlocked and available for input.

## 6. AN INOPERABLE OPERATOR CONSOLE:

Each time a write operation is initiated the time of the operation will be saved. Whenever ocpcm\_ is called -- for any reason -- it will check to see if a write operation has been in process for over a specified amount of time. This time limit will be 30 seconds. If the time limit for the write operation has been exceeded ocpcm\_ will assume that the operator console is now inoperable.

When ocpcm\_ determines that the console is inoperable it will try to issue another write command which will just turn on the beeper. It will also try to signal an initializer procedure. This procedure will try to write a message on some other terminal attached to the initializer process. This message will notify the operator of the condition of the operator console.

## 7. ASCII ←----→ GEBCD CHARACTER CONVERSION:

A consistent two-way transliteration between the ASCII and GEBCD character sets will be provided. Through the use of the escape character "\" every ASCII character will have an unambiguous GEBCD

representation. Messages written on the operator console may actually contain the escape character followed by the character to be interpreted or the octal number which represents this ASCII character. Table IV shows the BCD input needed to express each ASCII character and it shows the BCD output representation of each ASCII character.

TABLE I  
Format of syserr\_log header and entry

```

dcl 1 slog          based (slog_ptr) aligned, /* HEADER */
    2 lock          bit (36),                /* Locks the whole
                                                segment. */
    (2 last        bit (18),                /* Offset of last entry. */
     2 len          fixed bin (17))         /* Length of buffer in
                                                characters. */
    2 save (14)    bit (36),                /* Reserved for future
                                                use. */
    2 butter        char (slog.len),         /* Area containing
                                                message entries. */
    2 ext (8)      bit (36);                /* Dummy area. */

dcl 1 smess        based (smess_ptr) aligned, /*MESSAGE ENTRY */
    2 time          fixed bin (71),         /* Raw time message
                                                logged. */
    (2 prev        bit (18),                /* Offset of previous
                                                entry. */
     2 next        bit (18),                /* Offset of next entry. */
     2 seq_num     fixed bin (17),         /* Sequence number. */
     2 code        fixed bin (8),         /* syserr code. */
     2 len         fixed bin (8))         /* Length of text in
                                                characters. */
    2 text         char (smes.len)         /* ASCII message text. */
    2 end          char (1);                /* Dummy */

```

Note: These declarations can be found in ... syserr\_log.incl.pl1



**TABLE II**  
**The PART LOG Configuration Card:**

**PART LOG FREC(1)NREC(1) ... FREC (max id)NREC(max id)**

1. The term LOG identifies the partition.
2. FREC is the number of the first record assigned to the partition on the specified device.
3. NREC is the number of records assigned to the partition on the specified device.

TABLE III  
The LOG Configuration Card:

LOG	INIT	CELL	MODE	SYS_BUF
-----	------	------	------	---------

1. INIT is a flag that is equal to either 0 or 1.  
0 → The syserr\_log segment is not to be initialized.  
1 → Initialize the syserr\_log
  - a) Reset sequence number to zero.
  - b) Start next message at top of buffer. A dummy message containing the new sequence number will be created.
2. CELL is the interrupt cell number used for the log interrupt. It should be 23.
3. MODE is a flag that is equal to either 0 or 1.  
0 → The operator console should run in the SERVICE MODE.  
1 → It should run in the DEVELOPMENT MODE.  
Note, if this configuration card is missing the SERVICE MODE will be used.
4. SYS\_BUF should be a number from 1 to 14. It is used to determine the size of the syserr write buffer. The ocdcm\_write buffers are really one buffer. It contains room for 15 messages. This configuration parameter is used to divide this one buffer into a syserr part and a dim part. If this value is 12 then room for 12 syserr messages and 3 dim messages will be reserved.

Note, if this configuration card is missing the default value will be 10.

BCD → ASCII → BCD

BCD INPUT	ASCII	BCD OUTPUT
-----------	-------	------------

000	1000	1000
001	1001	1001
002	1002	1002
003	1003	1003
004	1004	1004
005	1005	1005
006	1006	1006
007	1007 BEL	1007
010	1010 BS	1010
011	1011 HT	1011
012	1012 NL	1012 CR
013	1013 VT	1013
014	1014 NP	1014
015	1015	1015
016	1016 RRS	1016
017	1017 BRS	1017
020	1020	1020
021	1021	1021
022	1022 HLF	1022
023	1023	1023
024	1024 HLR	1024
025	1025	1025
026	1026	1026
027	1027	1027
030	1030	1030
031	1031	1031
032	1032	1032
033	1033 mc	1033
034	1034	1034
035	1035	1035
036	1036	1036
037	1037	1037

BCD INPUT	ASCII	BCD OUTPUT
-----------	-------	------------

040	blank	blank	blank
041	!	!	!
042	"	"	"
043	#	#	#
044	\$	\$	\$
045	%	%	%
046	&	&	&
047	'	'	'
050	(	(	(
051	)	)	)
052	*	*	*
053	+	+	+
054	,	,	,
055	-	-	-
056	.	.	.
057	/	/	/
060	0	0	0
061	1	1	1
062	2	2	2
063	3	3	3
064	4	4	4
065	5	5	5
066	6	6	6
067	7	7	7
070	8	8	8
071	9	9	9
072	:	:	:
073	;	;	;
074	<	<	<
075	=	=	=
076	>	>	>
077	?	?	?

BCD INPUT	ASCII	BCD OUTPUT	BCD OUTPUT
100	/ @	@	@
101	/ A	A	A
102	/ B	B	B
103	/ C	C	C
104	/ D	D	D
105	/ E	E	E
106	/ F	F	F
107	/ G	G	G
110	/ H	H	H
111	/ I	I	I
112	/ J	J	J
113	/ K	K	K
114	/ L	L	L
115	/ M	M	M
116	/ N	N	N
117	/ O	O	O
120	/ P	P	P
121	/ Q	Q	Q
122	/ R	R	R
123	/ S	S	S
124	/ T	T	T
125	/ U	U	U
126	/ V	V	V
127	/ W	W	W
130	/ X	X	X
131	/ Y	Y	Y
132	/ Z	Z	Z
133	/ [	[	[
134	/ \	\	\
135	/ ]	]	]
136	/ =	=	=
137	/ ←	←	←

BCD INPUT	ASCII	BCD OUTPUT	BCD OUTPUT
140	/	-	-
141	/ A	a	A
142	/ B	b	B
143	/ C	c	C
144	/ D	d	D
145	/ E	e	E
146	/ F	f	F
147	/ G	g	G
150	/ H	h	H
151	/ I	i	I
152	/ J	j	J
153	/ K	k	K
154	/ L	l	L
155	/ M	m	M
156	/ N	n	N
157	/ O	o	O
160	/ P	p	P
161	/ Q	q	Q
162	/ R	r	R
163	/ S	s	S
164	/ T	t	T
165	/ U	u	U
166	/ V	v	V
167	/ W	w	W
170	/ X	x	X
171	/ Y	y	Y
172	/ Z	z	Z
173	/ [	[	[
174	/ ↑	↑	↑
175	/ ]	]	]
176	/ \$	\$	\$
177	/	~	~