From: Charles Hornig

To: MTB Distribution

Date: January 8, 1981

Subject: Redoing IOM Management (MTB-484)

This MTB contains a proposal to reimplement the supervisor's IOM management routines with new interfaces. Many events have combined to make this a good idea now.

First, support for the UNCP requires that we support "paged" mode on NSA IOM's. This is necessary for continued system security. If "paged" mode were not used, the UNCP would have access to all of Multics main memory through the DIA. This would give it access to read confidential information and to severely damage the system in the event of a failure. Since the UNCP software was developed outside the Multics project and can not be controlled by the requirements of Multics auditing, it is necessary to restrict its access to Multics. This will require significant changes to `iom_manager`.

Second, the Orion development effort requires IOX support software which is compatible with the existing IOM software so that its callers (`ioi_`, `ocdcm_`, IMP DIM, etc.) need not be modified for Orion. This means changing the subroutine call specifications for the current IOM software so that it can be compatible with the planned IOX code.

The Orion effort also involves recoding the non-critical parts of I/O management in PL/1 instead of ALM. This will improve the readability and maintainability of this subsystem. The critical paths (connecting, status fetching, and interrupts) will remain in ALM for speed.

Customers have requested that we support up to four IOM's on the DPS-8/M. This has never been done because it would involve a rewrite of `iom_manager`. Once the code is open, it is easy to do. Support for the IIOC can also be added in the same manner.

Finally, there are performance improvements in interrupt handling which may be made now that the Bulk Store is no longer supported. Since all interrupts are now from IOM's, it is possible to eliminate ii and have interrupts handled directly by the IOM code.

When support for IOM's which are incapable of "paged" operation is removed it will be possible to make several significant improvements to the I/O system. While I/O buffers will continue to be located in the low-order system controller, it will no longer be necessary for them to be allocated in contiguous pages in the low 256K. This will significantly decrease the cost of using large buffers. "Paged" mode will also make possible improved disk DIM performance.

There is enough to be gained from a complete rewrite to make the effort worthwhile. This will provide us with an I/O management subsystem in accordance with current Multics programming standards and supporting hardware currently unsupportable.

---

io←manager

io_manager and io_manager_wired are the supervisor's interface to the I/O hardware. Entries are provided to assign and use I/O channels. The entries in io_manager may be called on unwired stacks. Those in io_manager_wired must be called with interrupts masked on a wired stack. Many functions are common to both.

io←manager$assign

This entry is used to assign an I/O channel. A channel must be assigned before it may be used.

Usage:

```
dcl io_manager$assign entry (bit (36) aligned, fixed bin (3), fixed bin (8),
       entry (bit (72) aligned, fixed bin (3), bit (36) aligned),
       bit (72) aligned, fixed bin (35));

call io_manager$assign (chx, iom, channel, handler, value, code);
```

where:

1. chx — is the unique index associated with this I/O channel. This value is used in all future calls to io←manager. (Output)

2. iom — is the IOM/IOX number of the channel to be assigned. It may range from one to four. (Input)

3. channel — is the channel number of the channel to be assigned. It may range from 0 to 63. (Input)

4. handler — is the interrupt handler to be associated with the channel. It will be invoked on a wired stack when an interrupt is received from the channel. Its calling sequence is:

```
dcl handler variable entry (bit (72) aligned, fixed bin (3),
       bit (36) aligned);

call handler (value, level, word);
```

where:

1. value — is the index value passed to io←manager$assign. (Input)

2. level — is the interrupt level signalled. It may be 1, 3, 5, or 7. (Input)

3. `word` is a data word associated with a level 1 (system fault) or level 7 (special status) interrupt. It is set to ""b for level 3 (terminate) and level 5 (marker) interrupts. (Input)

5. `value` is an index value to be passed to the interrupt handler. It may be used by the interrupt routine to locate its own data bases. (Input)

6. `code` is a standard system status code. (Output)

io←manager$set←unqueued←status

This entry sets up basic status reporting for a channel. It terminates the effect of a previous call to `io_manager$set_status_queue`. This entry must be called before any status can be obtained.

Usage:

```
dcl io_manager$set_unqueued_status entry (bit (36) aligned, ptr,
      fixed bin (35));

call io_manager$set_unqueued_status entry (chx, statusp, code);
```

where:

1. `chx` is a channel index. (Input)

2. `statusp` is a returned pointer to the word in which marker and terminate status will be stored by the hardware. (Output)

3. `code` is a standard system status code. (Output)

io←manager$set←status←queue

This entry allows the caller to define a circular queue in which marker and terminate status and extended status will be stored. It undoes the effect of a previous call to io←manager `set_unqueued_status`.

Usage:

```
dcl io_manager$set_status_queue entry (bit (36) aligned, ptr, fixed bin (35));

call io_manager$set_status_queue (chx, statusqp, code);
```

where:

1. `chx` is a channel index. (Input)
2. `statusqp` is a pointer to an `io_status_queue` structure. (Input)
3. `code` is a standard system status code. (Output)

Notes:

The `io_status_queue` structure is described by `io_status_queue.incl.pl1`.

io←manager$unassign

This entry terminates the effect of a previous call to `io_manager$assign`.

Usage:

```
dcl io_manager$unassign entry (bit (36) aligned, fixed bin (35));

call io_manager$unassign (chx, code);
```

where:

1. `chx` is a channel index. It is reset to an invalid value to prevent re-use. (Input/Output)
2. `code` is a standard system status code. (Output)

io←manager$set←list←abs
io←manager←wired$set←list←abs

This entry or one of the other "set←list" entries must be called before calling any of the "connect" entries. This entry sets up information for absolute or extended I/O addressing.

Usage:

```
dcl io_manager$set_list_abs entry (bit (36) aligned);
dcl io_manager_wired$set_list_abs entry (bit (36) aligned);

call io_manager$set_list_abs (chx);
call io_manager_wired$set_list_abs (chx);
```

where:

1. `chx` is a channel index assigned by `io_manager$assign`. (Input)

io←manager$set←list←rel
io←manager←wired$set←list←rel

This entry sets up information for future connects in relative mode.

Usage:

```
dcl io_manager$set_list_rel (bit (36) aligned, ptr, fixed bin (19));
dcl io_manager_wired$set_list_rel (bit (36) aligned, ptr, fixed bin (19));

call io_manager$set_list_rel (chx, basep, bound);
call io_manager_wired$set_list_rel (chx, basep, bound);
```

where:

1. `chx` is a channel index. (Input)

2. `basep` is a pointer to the base of the workspace to be defined for the channel. This must point to a contiguous abs-wired segment in the low 256K of memory. (Input)

3. `bound` is the size of the workspace in words. It must be a multiple of 512 words. `basep` and `bound` define the area accessible by the channel in relative mode. (Input)

io←manager$set←list←paged
io←manager←wired$set←list←paged

This entry entry sets up for future connects in paged mode.

Usage:

```
dcl io_manager$set_list_paged entry (bit (36) aligned, ptr, fixed bin (19));
dcl io_manager_wired$set_list_paged entry (bit (36) aligned, ptr,
       fixed bin (19));

call io_manager$set_list_paged (chx, ptp, bound);
call io_manager_wired$set_list_paged (chx, ptp, bound);
```

where:

1. `chx` is a channel index. (Input)

2. `ptp` is a pointer to an IOM page table describing a segment. This page table will be used by the IOM in future paged I/O operations. The page table must be contiguous and abs-wired in the low 256K of memory. (Input)

3. `bound` is the number of words in the I/O segment defined by the page table. (Input)

io←manager$connect←abs
io←manager←wired$connect←abs

This entry causes a connect to be sent to the channel in absolute mode. One of the "set←list" entries must have been called previously. The DCW list may switch to relative or paged mode if the proper "set←list" entry has been called.

Usage:

```
dcl io_manager$connect_abs entry (bit (36) aligned, bit (36) aligned, ptr);
dcl io_manager_wired$connect_abs entry (bit (36) aligned, bit (36) aligned,
       ptr);

call io_manager$connect_abs entry (chx, pcw, listp);
call io_manager_wired$connect_abs entry (chx, pcw, listp);
```

where:

1. `chx` is a channel index. (Input)
2. `pcw` is the first word of the PCW to be sent to the channel. (Input)
3. `listp` points to the DCW list to be executed by the channel. It should be null if no DCW list is supplied. (Input)

io←manager$connect←rel
io←manager←wired$connect←rel

This entry causes a connect in relative mode to be sent to the channel. `io_manager$set_list_rel` must have been called previously.

Usage:

```
dcl io_manager$connect_rel entry (bit (36) aligned, bit (36) aligned,
       fixed bin (18))
dcl io_manager_wired$connect_rel entry (bit (36) aligned, bit (36) aligned,
       fixed bin (18))

call io_manager$connect_rel (chx, pcw, listx);
call io_manager_wired$connect_rel (chx, pcw, listx);
```

where:

1. chx — is a channel index. (Input)
2. pcw — is the first word of the PCW to be sent to the channel. (Input)
3. listx — is the zero-origin offset in the workspace of the beginning of the DCW list. (Input)

io←manager$connect←paged
io←manager←wired$connect←paged

This entry causes a connect in relative mode to be sent to the channel. io_manager$set_list_paged must have been called previously.

Usage:

```
dcl io_manager$connect_paged entry (bit (36) aligned, bit (36) aligned,
       fixed bin (18))
dcl io_manager_wired$connect_paged entry (bit (36) aligned, bit (36) aligned,
       fixed bin (18))

call io_manager$connect_paged (chx, pcw, listx);
call io_manager_wired$connect_paged (chx, pcw, listx);
```

where:

1. chx — is a channel index. (Input)
2. pcw — is the first word of the PCW to be sent to the channel. (Input)
3. listx — is the zero-origin offset in the workspace of the beginning of the DCW list. (Input)

io←manager←wired

These entries are used during system inititalization and shutdown.

io←manager←wired$ignore←interrupt

This entry may be used in calls to io_manager$assign if interrupts should be ignored. It simply returns.

Usage:

```
dcl io_manager_wired$ignore_interrupt entry (bit (72) aligned, fixed bin (3),
       bit (36) aligned));

call io_manager$assign (chx, iom, channel, io_manager_wired$ignore_interrupt,
       ""b, code);
```

io←manager←wired$reset

This entry is called by emergency_shutdown to reset some io_manager databases.

Usage:

```
dcl io_manager_wired$reset entry ();

call io_manager_wired$reset ();
```

io←manager←wired$run

This entry causes the IMW area to be polled for pending interrupts. It may be used while interrupts are masked or otherwise disabled to cause status processing to occur and interrupt handlers to be called.

Usage:

```
dcl io_manager_wired$run entry ();

call io_manager_wired$run ();
```

```
/* Begin include file io_status_queue.incl.pl1 */

dcl io_status_queue_version_1 char (4) aligned static options (constant) init ("isq1");

dcl io_status_queue_ptr ptr;
dcl 1 io_status_queue aligned based (io_status_queue_ptr),
      2 header,
        3 version char (4),
        3 entry_count fixed bin,                          /* number of queue entries */
        3 current_entry fixed bin,                        /* oldest unread entry */
        3 next_entry fixed bin,                           /* where to put next status */
        3 pad (12) bit (36),
      2 status (0 refer (io_status_queue.entry_count)),
        3 word1,
          4 in_use bit (1) unaligned,                     /* this entry contains valid status */
          4 power_off bit (1) unaligned,                  /* DC power off */
          4 major_status bit (4) unaligned,               /* device major status */
          4 substatus bit (6) unaligned,                  /* device substatus */
          4 odd_word bit (1) unaligned,                   /* */
          4 marker bit (1) unaligned,                     /* this is marker status */
          4 rfu1 bit (2) unaligned,
          4 initiate_interrupt bit (1) unaligned,         /* */
          4 rfu2 bit (1) unaligned,
          4 channel_status bit (3) unaligned,             /* status from channel */
          4 central_status bit (3) unaligned,             /* status from I/O central */
          4 action_code bit (2) unaligned,                /* */
          4 write bit (1) unaligned,                      /* */
          4 character_position uns fixed bin (3) unaligned,
          4 record_residue uns fixed bin (6) unaligned,
        3 word2,
          4 next_lpw_offset bit (18) unaligned,           /* 1 + offset of last DCW */
          4 rfu bit (18) unaligned,
        3 word3 bit (36),
        3 word4,
          4 next_dcw_offset bit (18) unaligned,           /* 1 + offset of last data word */
          4 character_position uns fixed bin (3) unaligned,
          4 write bit (1) unaligned,
          4 action_code bit (2) unaligned,
          4 tally_residue uns fixed bin (12) unaligned,   /* residue of last DCW tally */
        3 word5,
          4 data_count uns fixed bin (26) unaligned,      /* total words transferred (on IIOC & IOX) */
          4 rfu bit (10) unaligned,
        3 words (6:16) bit (36);

/* End include file io_status_queue.incl.pl1 */
```