To: Distribution

From: Noel I. Morris & Bernard S. Greenberg

Date: May 7, 1974

Subject: I/O Buffer Management in Multics

Part I: Introduction

Overview

In the design documentation for the I/O Interfacer (MTB-028 and MTB-056), methods are discussed for the proper handling of I/O buffers. From these design considerations, it became clear that a specialized interface was needed between the I/O Interfacer and Multics page control. The I/O Buffer Manager is designed to be such an interface. It is responsible for maintaining the workspace buffer segment required by each user of the I/O Interfacer. The I/O Buffer Manager must ensure that the workspace buffer is wired-down prior to performing I/O. It must take steps to unwire the buffer after I/O completion. If the buffer is longer than 1024 words (one page), it must ensure that all pages are wired-down contiguously.

Abs-usable Memory

During Multics operation, the low-order system controller is never subject to dynamic deconfiguration. The core storage contained in this system controller is referred to as abs-usable memory. Frames of abs-usable memory may be temporarily wired for use as I/O buffer workspace. No consideration need be given to the problems of dynamically removing a system controller which contains wired I/O buffers. (A frame is a 1024 word block of core beginning at a 0 mod 1024 address. A frame can be occupied by a Multics page.)

The I/O Buffer Manager will, through calls to page control, be able to free up a frame in abs-usable memory and to assign a page of an I/O workspace buffer segment to that frame. The I/O Buffer Manager must take special steps when a buffer is larger than one page. In this case, contiguous frames must be freed and then assigned to the workspace buffer. Note that pages of the workspace buffer will be wired into abs-usable core only when I/O is actually taking place. At other times, the buffer will be unwired, and the core frames that it occupied will be available for normal Multics paging.

Part II: Buffering by the I/O Interfacer

Creating an I/O Workspace Buffer Segment

The I/O Interfacer will be responsible for creating a workspace buffer segment for each assigned device. This segment can be created in the caller's process directory. The ring brackets of the segment will be set, based on the current validation level, to (v, v, v) with rw access only to the caller's process. This segment is writeable in rings 0 through v.

At this time, the segment must be made active. That is, the segment must have an entry in the active segment table (an AST entry) and a page table. If the entry hold switch in an active segment's AST entry is turned on, that segment will be guaranteed not to be deactivated. A Multics system primitive will be available to turn on the entry hold switch in a segment's AST entry and return a pointer to the AST entry. The I/O Interfacer must save the AST entry pointer for later use. Once the workspace buffer segment has been created and activated, it can be used to set up IOM DCW lists and data. It is not wired-down at this time, however.

Wiring an I/O Workspace Buffer

When I/O is to be performed on an I/O workspace buffer, the I/O Buffer Manager must be called. The I/O Buffer Manager will grab as many frames as needed for the workspace buffer from available frames in abs-usable memory. It will make the appropriate calls to page control to flush the previous contents of these frames. Then it will assign the pages of the workspace buffer to these frames contiguously, calling page control to wire each page of the workspace buffer into its assigned frame. I/O can now be performed using the workspace buffer.

Unwiring an I/O Workspace Buffer

When I/O has terminated on a device, the workspace buffer can be unwired. The I/O Buffer Manager will be called at I/O termination. However, the pages of the workspace buffer will not actually be unwired at this time. An active device will probably receive another connect shortly after a terminate. It is extremely inefficient to unwire the workspace buffer only to have to wire it again almost immediately. Therefore, the I/O Buffer Manager will "remember" that I/O has terminated on a particular workspace buffer, and, after an appropriate time interval has elapsed with no further connects, will then actually unwire the buffer. This mechanism will avoid unnecessary calls to page control. It will also avoid tying up wired-down core for idle devices which remain attached.

A time-out entry in the I/O Buffer Manager will be called by the Multics traffic controller at appropriate time intervals to perform the actual unwiring of idle workspace buffer pages. The time interval must be short enough to not tie up core unnecessarily. Yet, it must be long enough to prevent needless unwiring and rewiring of pages. A time interval of between 10 and 30 seconds will most likely be acceptable.

Deleting an I/O Workspace Buffer Segment

When a device is unassigned, the I/O workspace buffer segment must be deleted. First, the I/O Interfacer must take steps to ensure that no I/O is still taking place. Then, an entry in the I/O Buffer Manager must be called to force unwire all pages of the workspace buffer. Upon return from the I/O Buffer Manager, the workspace buffer segment can be deactivated and then deleted.

Note that care must be taken to detect that the workspace buffer is still wired-down when the force unwire entry is called. If the buffer has already been unwired, the force unwire entry must just return to its caller. The I/O Buffer Manager must be extremely careful not to unwire pages that no longer belong to the workspace buffer and might, in fact, belong to another buffer.

Changes to I/O Interfacer Specifications

The scheme described above completely replaces the abs-wireable bit in the AST entry as designed in MTB-056. The page control modifications described in that document need not be implemented. In addition, the workspace buffer size limitation described in MTB-056 will not exist.

Part III: Buffering by Hardcore Users of the I/O Interfacer

A Hardcore Input Buffering Strategy

Designers of DIMs which reside in the hardcore should also take advantage of the facilities provided by the I/O Buffer Manager. One buffering scheme (proposed by R. Kanodia for the ARPA Netowrk IMP DIM) would allow a semi-infinite number of large segments to be filled up, one block at a time, by input from a device. As each block became full, the DIM would, at interrupt time, initiate I/O on the next block. When a segment became full, the DIM would, again at interrupt time, switch to the next sequential segment. Outer ring callers to such a DIM could pick up their data at their own convenience. When all data from an input segment was picked up, the segment could be deleted by the DIM. Only the block currently accepting input would be wired-down.

Sharing Non-hardcore Segments

The strategy described above requires that the buffer segment in use be available in the address space of every process running under Multics, since an I/O completion interrupt can occur in any running process. Normally, only hardcore segments, loaded from the Multics system tape during system initialization, can be shared with the same segment number among many processes. But, the scheme above may require a rather large number of buffer segments, and there is a definite limit to the number of haardcore segments which may be loaded during system initialization. Therefore, an alternative method is proposed below to allow the sharing of buffer segments:

The "abs seg"

A program running in Multics hardcore has access to the descriptor segment of the process in which it is running. It is therefore possible for a hardcore program to construct a segment descriptor word (SDW) pointing to any legitimate page table, to store it somewhere in the descriptor segment, and to reference the pages pointed to by that page table. This can be done through the normal use of ITS pointers referencing the segment number which would cause that manufactured SDW to be used during 6180 address preparation. Note that the entire access control mechanism can be bypassed in this way.

In fact, several hardcore programs in Multics do use this method to access segments not normally in their address space. The way in which this is accomplished is to load a zero length, unpaged segment during Multics initialization. Although no data is actually loaded, a word-pair is reserved in the hardcore portion of the descriptor segment of all processes. The segment names supplied from the Multics system tape are placed in the

Segment Loading Table, and hardcore segments can refer to this "fictitious" segment by name. During system operation, the reserved word-pair can be filled in with any manufactured SDW when needed. Once this has been done, the segment name and number of this "fictitious" segment refer, in this process, to the segment actually described by the manufactured SDW. Such a "fictitious" segment is generally referred to as an "abs_seg".

Buffer Segment Creation and Management

The continuous, semi-infinite buffer strategy described above can easily be implemented by a hardcore DIM through the use of an "abs_seg" as described. The DIM must always have already created the next buffer segment in line to be used well before it is actually needed. This can be done via a call to append_branch. In order for a segment to be referenced through the use of an "abs_seg", the segment must be active. Thus, the segment must be activated, and the entry hold switch in its AST entry turned on.

When the DIM is ready to use the buffer segment, a system primitive can be called to construct an SDW given an AST entry pointer. This SDW can be deposited in the descriptor segment entry for an "abs_seg" reserved for use by this DIM. When I/O is to be performed, a special entry in the I/O Buffer Manager can be called to wire pages of the buffer. This entry can be called at interrupt time to wire null pages. When I/O completes, the standard unwire entry in the I/O Buffer Manager can be called to unwire the buffer pages. Note that the DIM, itself, is responsible for making these calls.

When a hardcore DIM is finished performing I/O on a segment, a system primitive must be called to turn off the entry hold switch in the segment's AST entry. The segment can still be referenced, but not for the purposes of performing I/O. Note that when use of an "abs_seg" is completed, it is customary to zero the word-pair in the descriptor segment containing the manufactured SDW.

Actions Taken at Interrupt Time

When an I/O completion interrupt takes place, the hardcore DIM can unwire the currently wired block of the buffer segment through a call to the I/O Buffer Manager. Then, the next block of the buffer can be wired. Note that when using the special wiring entry in the I/O Buffer Manager, all pages to be wired must be null. The system is incapable of waiting for a page to be read in from secondary storage at interrupt time. Once the next block of the buffer is wired, a call can be made to ioi_$hardcore_workspace and a DCW list can be constructed in the new block. Then, a call can be made to ioi_$connect. This

completes the interrupt processing.

When a particular buffer segment becomes filled, buffer segments can be swapped at interrupt time. The DIM can construct an SDW pointing to the next buffer segment from the segment's AST entry pointer. (Remember that the DIM had created and activated this next segment in advance.) This new SDW can now be used in conjunction with the DIM's "abs_seg".

Note that before a buffer segment can be referenced at interrupt time, it must be made available in the address space of the process that was interrupted. Therefore, the DIM's interrupt procedure must always fill in the SDW for its "abs_seg" before referencing the buffer segment.

Part IV: I/O Buffer Manager Calls

Entry: iobm$wire_buffer

This entry is called to wire-down pages of an I/O buffer segment. All wired pages will be guaranteed to reside in contiguous core frames. The absolute address of the base of the workspace buffer will be returned to the caller.

Usage

```
declare iobm$wire_buffer entry (ptr, fixed bin(18),
        fixed bin(18), fixed bin(24), fixed bin(17),
        fixed bin(35));
call iobm$wire_buffer (astep, base, nwords, absaddr,
        rqindex, rcode);
```

astep	is a pointer to the buffer segment's AST entry. (Input)
base	is the offset of the first location in the buffer segment to be wired. (Input)
nwords	is the number of words of the workspace buffer to be wired-down. (Input)
absaddr	is the absolute address of the base of the wired-down workspace buffer. (Output)
rqindex	is a request index. It must be used in later calls to the I/O Buffer Manager to identify a given request. The first time iobm$wire_buffer is called, it must be set to zero. (Input & Output)
rcode	is an error code. (Output)

Entry: iobm$wire_buffer_interrupt

This entry is identical to iobm$wire_buffer, except that it is intended for use by hardcore callers of the I/O Interfacer and may be called at interrupt time. There is one restriction in the use of this call: All pages to be wired-down must be null since this entry will be incapable of paging in non-null pages from secondary storage. An improper call to iobm$wire_buffer_interrupt will result in a non-zero error code being returned.

Usage

```
declare iobm$wire_buffer_interrupt entry (ptr, fixed
        bin(18), fixed bin(18), fixed bin(24), fixed
        bin(17), fixed bin(35));
call iobm$wire_buffer_interrupt (astep, base, nwords,
        absaddr, rqindex, rcode);
```

Entry: iobm$unwire_buffer

This entry will be called on I/O termination. Actual unwiring of the pages of the I/O buffer segment will not take place at this time. Flags will be set for each of the pages to cause them to be unwired later.

Usage

```
declare iobm$unwire_buffer entry (fixed bin(17), fixed
          bin(52));
call iobm$unwire_buffer (rqindex, delta);
```

rqindex is the request index returned in the call to iobm$wire_buffer. (Input)

delta is a time delta. After this time has elapsed, if no further I/O has been initiated using this buffer segment, the pages of the buffer segment will be unwired. (Input)

Entry: iobm$force_unwire_buffer

This entry is be called by the I/O Interfacer or a hardcore DIM prior to deleting a workspace buffer segment. It is the responsibility of the caller to ensure that all I/O involving this buffer has terminated.

Usage

```
declare iobm$force_unwire_buffer entry (ptr);
call iobm$force_unwire_buffer (astep);
```

astep is a pointer to the buffer segment's AST entry. (Input)

Entry: iobm$time_out

This entry is called by the Multics traffic controller at approproiate time intervals. It will check all wired workspace buffer pages to see if they can be unwired at this time.

Usage

```
declare iobm$time_out entry;
call iobm$time_out ();
```

(There are no arguments.)

Entry: grab_aste$grab_aste_io

This entry will ensure that a segment is active. Given a pointer to a segment, it will return a pointer to that segment's AST entry. The entry hold switch in the AST entry will be on.

Usage

```
declare grab_aste$grab_aste_io entry (ptr, fixed
          bin(35)) returns (ptr);
astep = grab_aste$grab_aste_io (segptr, rcode);
```

segptr — is a pointer to the segment to be made active. (Input)

astep — is a pointer to the segment's AST entry. (Output)

Entry: grab_aste$release_io

This entry is called to allow an I/O buffer segment to be deactivated. The entry hold switch in the segment's AST entry is turned off.

Usage

```
declare grab_aste$release_io entry (ptr, fixed
          bin(35));
call grab_aste$release_io (astep, rcode);
```

astep — is a pointer to the AST entry of a buffer segment. (Input)

Part V: I/O Buffer Manager Implementation

The Interface to Page Control

The I/O buffer manager provides the ability to abs-wire contiguous pages of arbitrary segments into contiguous core frames at arbitrary (i.e., possibly interrupt) times. In order to wire pages, in general, page reading from secondary storage must be performed, and waiting for the completion of such reading. As the latter is impossible at interrupt time, the restriction is made that only null pages may be abs-wired at interrupt time. Only the entry iobm_$wire_buffer_interrupt is permitted such calls.

The wiring of pages on behalf of the I/O Buffer Manager wiring entries normally involves the contiguous allocation of abs-usable memory. However, if the pages to be wired have already been abs-wired by this mechanism (i.e., a previous unwire call has not timed out), no allocation or wiring is necessary. If allocation and wiring are necessary, though, the core map will be scanned for the first contiguous block of abs-usable core frames large enough to meet the request. Obviously, none of the pages in this block can already be abs-wired. The occupants of these core frames will be forcibly evicted. The pages of the segment which are being requested to be wired are then read into these core frames in the correct order, and the core frames are marked as being abs-wired. Note, that if the pages to be read are null, no page need be read in and no waiting need be done.

If no contiguous block of abs-usable core frames large enough to satisfy the request is found, a currently non-abs-usable system controller will be made abs-usable, preventing it from being dynamically deconfigured. However, a current hardware restriction in the IOM forces DCW lists for data channels to be in the first 256K of memory, and thus, only this much memory can be comandeered for abs-wiring in this way.

Unwire calls to the I/O Buffer Manager involve no interaction with page control. A timer is set from the parameter "delta" (see Bookkeeping), and control returned. Upon the maturation of this timer, if an intervening wire call to the same pages has not been made, these pages will be unwired, and the core marked as no longer abs-wired. At the time of the unwire timeout, we assume that the contents of these pages may have changed due to I/O. Thus, we set the page-has-been-modified bit in the page table words for each of these pages before unwiring.

The Interface to Segment Control

It is essential to prohibit I/O buffer segments from being deactivated while they are being used as I/O buffers. Doing this requires setting the entry hold active (aste.ehs) switch in the

AST entry of the segment concerned. This implies, for one, that the segment has an AST entry. Hence, a primitive is provided for use by the I/O Interfacer, grab_aste$grab_aste_io, which, given segment pointer, returns an AST entry pointer. This pointer points to an AST entry whose entry hold switch has just been turned on. Activation will be accomplished by locking the dirctory containing the segment of interest, and then taking a segment fault on the segment. The entry hold switch will be turned on before the directory is unlocked. This insures that the segment will not be deactivated, and that its page table and AST entry will not move until the entry hold switch is turned off.

This entry, grab_aste$grab_aste_io, will also inform cache control that the segment being used as an I/O buffer is not to be encacheable in any process, and must be driven out of caches now. A special bit will be set in the AST entry of the segment to this effect, and SDW's will be modified.

At the time that a hardcore DIM or the I/O Interfacer is finished using a given segment as an I/O buffer, the cache inhibit and entry hold switches should be turned off. The grab_aste$release_io entry is provided for this purpose. Prior to calling this entry, however, the user of the buffer segment should call iobm$force_unwire to clean up the data bases of the I/O Buffer Manager with respect to this buffer segment.

Bookkeeping

The I/O Buffer Manager needs a table to keep track of various wire and unwire requests. Furthermore, coordination with the traffic controller is required in order to perform unwire timouts. A table will be set up in a hardcore segment with the following format:

```
declare 1 iobm_data$ external aligned,
          2 lock bit(36),
          2 free_queue bit(18) unaligned,
          2 in_use_queue bit(18) unaligned,
          2 timing_out_queue bit(18) unaligned,
          2 last_allocated fixed bin(17) unaligned,
          2 pad bit(36) unaligned,
          2 requests (1:63),
            3 unwire_time fixed bin(52),
            3 astep bit(18) unaligned,
            3 fp bit(9) unaligned,
            3 np bit(9) unaligned,
            3 thread bit(18) unaligned,
            3 flags,
              4 used bit(1) unaligned,
              4 timing_out bit(1) unaligned;
```

Three lists of "request" entries are maintained. The list heads are kept in "free_queue", "in_use_queue", and "timing_out_queue". The free queue is a list of unallocated entries. These entries have their "used" bits off. The in-use queue is a list of entries representing wire requests for which no unwire request has been received. These entries have their "used" bits on and their "timing_out" bits off. The timing-out queue is a list of entries representing wire requests for which unwire requests have been made, but has not yet timed out. These entries have their "used" and "timing_out" bits on, and "unwire_time" contains the clock time after which they should be unwired. The thread is in order of increasing unwire time. Note that neither the "used" nor "timing_out" bits are strictly necessary, due to the list structure, but should be useful as diagnostic aids. The "thread" field maintains these lists. This entire table is protected by a loop-type lock. No page faults or interrupts may be taken while the lock is set.

The fields "astep", "fp", and "np" in the entry represent the relative AST entry pointer, the first page number, and the number of pages of the segment wired by the request.

On any wire request, the first order of business is to check if this request is already satisfied by a previous request which has not yet timed out. The data table is locked. The request entry corresponding to the request index supplied by the caller (if nonzero) is inspected. If the entry agrees with the arguments given by the caller (i.e., the fields "astep", "np", and "fp" in the request entry are identical to the values computed from the caller's arguments), and the entry is in the timing-out queue, it is threaded out of this queue and into the in-use queue. The pages represented by the entry remain abs-wired, the table is unlocked, and control returned to the caller. If the entry agress with the values supplied in the call, but it is in the in-use queue, or if it is inconsistent with the supplied values and is in the timing-out queue (i.e., pages are shared but not coincident), the tables are unlocked and an error is returned. If the entry is free or does not agree with the call, a new entry is allocated, and the caller's request index set to the index of this new entry. The entry is then threaded into the in-use queue, the table is unlocked, and the page tables are locked. The requested pages are abs-wired into the first contiguous block of abs-usable memory large enough to satisfy the request. The page tables are temporarily unlocked while the process waits for pages to be physically read from secondary storage. The page tables are unlocked and control returned to the caller when all pages are wired.

At unwire call time, the caller's request index is assumed to be correct. The data table is locked. The request entry, which should be in the in-use queue, is threaded into the correct place in the timing-out queue, and the "unwire_time" in it is set to the current clock time plus the given delta. The data table

is unlocked, and control is returned to the caller.

A variable is kept which contains the "unwire_time" of the head of the timing-out queue, or a very large number if the queue is empty. The traffic controller polling mechanism will call iobm$time_out if the clock passes this time and the data table is not already locked. That is, if the data table is already locked, iobm$time_out will not be called. At that time, the following conditions exist: The traffic controller is unlocked, the AST is not lockable, nor are directories, and page faults are prohibited. The page tables, however, are lockable. When this entry is called, the clock is checked against the unwire time in the head of the timing-out queue. If the clock is less than the time in this entry, iobm$time_out returns. Otherwise, all entries in the timing out queue whose unwire time has passed are processed and freed. The variable interrogated by the traffic controller is reset to the time in the new entry at the head of the timing-out queue. The page tables are locked, the relevant pages unwired, the page-has-been-modified bit set in each PTW, and the pages tables unlocked.

The iobm$force_unwire entry scans all timing-out queue entries for requests involving the segment whose AST entry pointer was passed in the call. All such entries are forcibly timed-out at this time, unwiring memory and freeing request entries. The in-use queue is also scanned. If any entries corresponding the this buffer segment are found, an error is indicated.