Chapter 12
Memory Manager

The new version of the memory manager has quite a few new features to improve overall system performance. For users of the new Discovery ROM we have added an Improved memory peeker that can be interupted with the escape key during a listing. For Universe and discovery users we have added a faster memory search that allows allocation of handles to be much faster. This new scheme remembers the last handle allocated and starts searching for free memory after that handle the next time newHandle is called. And finally we have added an out of memory queue (OOMqueue) for improved application memory management.

OutOfMem Queue

When the memory manager cannot make a handle with memory in its current configuration it calls an internal routine that will do the following.

1. Run each entry in the OOMQueue until one of the routines reports that it has freed enough memory. If this happens the memory manager will try to allocate the new handle again.
2. Compact memory and try allocation again.
3. Purge Level 3 handles, if enough memory has been freed, compact memory and try allocation again.
4. Purge Level 2 handles if enough memory has been freed, compact memory and try again.
5. Purge Level 1 handles if enough memory has been freed, compact memory and try again.
6. Run the OOMQueue if the user freed enough memory, purgeall purgables, compact memory and try allocation for the last time.

If after all this the memory manager can still not allocate enough memory, it will report out of memory to the application.

Any applicaiton/DA/Init that installs itself in the OOM Queue is responsible for de-installing itself before it leaves memory. OOM Queue routines are installed and removed with AddToOOMQueue and RemoveFromOOMQueue. The OOM routine starts with a header record followed directly by its code, the header record looks like this:

Reserved	LONG	used as link to next task in queue
Version	WORD	Must be zero. Indicates what kind of OOM routine this is. Lets us change inputs in future.
Signature	WORD	$A55A signature (like heartbeat task header)
CodeEntry		your code starts right here

When an OOM routine is run, the memory manager pushes space for a long output on the stack, pushes two inputs on the stack and "jsl"s to the CodeEntry point. The OOM routine is responsible for stripping the inputs off the stack and returning in the result long, the amount of memory freed.

If the Memory Manager finds that one of the Queue routines is not valid (the signature is $A55A), a system death will occur. The new system death error code is $0209.

When the OOM routine gets control the stack will look like:

Space	LONG	Space for resulting long
BytesNeeded	LONG	# Bytes needed by memory mgr

Stage	WORD	word indicating the stage of the memory failure. 0=before memory manager tried to do anything. 1=after memory manager tried to do anything.
RTLAdr	3 bytes	return address

Just before the the OOM routine returns to the memory manager thw stack should look like:

AmountFreed	LONG	Number of bytes freed
RTLAdr	3 bytes	return address

While your routine is running it may call any memory manager routines it wishes. When calling any routine that might allocate more memory (like NewHandle) you must insure that you have enough memory for the new handle before you make the call. For instance, if you want your out of memory routine to save some data to disk before you purge it, be sure that you keep in mind that openning a file will cause memory allocation, and if you do not have enough memory to perform that allocation, you might end up having your routine called infinitely.

AddToOOMQueue **Call $0C02**

Input
HeaderPtr LONG

Output
None

Adds the indicated routine to the out of memory queue. The input should be a pointer to the out of memory queue header at the start of your data space.

Possible Errors:

InvalidTag the $A55A signature was not where it was supposed to be, or is not present

DeleteFromOOMQueue **Call $0D02**

Input
HeaderPtr LONG

Output
None

Removes the indicated routine from the out of memory queue.

Possible Errors:

NotInList the pointer you passed is not a valid queue entry.
InvalidTag the $A55A signature was not where it was supposed to be, or is not present

An OOMqueue example

The following is an example out of memory queue routine that you might use in your application. The basic idea of this routine is that most times when an application runs out of memory, it may still want to perform some actions that require memory, like notifying the user of the low memory situation with a dialog box and then giving the user the oportunity to save their work. To implement a friendly low memory system you might try the following idea, when your application starts up, it should allocate a handle with enough reserve memory to complete most common operations, put up a dialog box, and save the work. In addition to all of this, you might want to have a flag in your data storage area that you can set to inform your event loop that its time to tell the user to leave.

This first routine is an example of how to install the OOMQueue routine (as well as allocating the reserve memory).

```
; first allocate a handle with enough memory for our low memory exit
; this example will use a 16k handle

                pha                     ; room for result
                pha
                PushLong #$4000         ; size of handle
                PushWord MyID           ; my applications ID
                PushWord #0             ; no bits set, unlocked and moveable
                PushLong #0             ; address (Not used)
                _NewHandle
                PullLong ResvHand       ; and pull off the reserve handle

                PushLong #MyOOMRtn      ; address of the OOM header
                ldx #$0C02              ; function number for AddToOOMQueue
                jsl >$E10000            ; and call the tool dispatcher

                stz OOMFlag             ; zero our low memory indicator
```

The following is the actual OOMqueue entry itself. It has been written for the MPWIIgs assembler.

```
; This is the OOMQueue header for our routine
MyOOMRtn        Record
                ds.L 0                  ; used by queue manager
                dc.W 0                  ; OOMEntry version
                dc.W $A55A              ; queue entry signature
                EndR
;
; Now for my out of memory routine
MyOOM           proc
; first set up the equates for the stack frame passed to us by the memory mgr
RTLAdr          equ 1                   ; return address we will go back to
Stage           equ RTLAdr+3            ; indicates when called
BytesNeeded     equ Stage+2             ; number of bytes the mem mgr needs
Result          equ BytesNeeded+4       ; return number of bytes freed
;
; before we start we should zero out the result
                lda #0
                sta Result,s            ; zero the result on the stack
                sta Result+2,s

; Since this routine can be called before and after purging data
; we want to wait till the memory manager has purged everything it can
; before we panic so the first thing we do is test the Stage
                lda Stage,s             ; get the passed stage
                beq OOMEnd              ; if 0 then don't free anything
```

```
; Now that we know that the memory manager has tried everythign else, we test
; to see if we have done this before by testing the OOMFlag
                lda >OOMFlag            ; must use long address DB=unknown
                bne OOMEnd              ; if non-zero then memory already free

; since we know that we have not freed the reserve memory yet, we will do so
; now and set the flag.
                PushLong >ResvHand      ; handle to our reserve space
                _DisposeHandle          ; and dispose of it

                lda #$FFFF              ; now set our flag to true
                sta >OOMFlag            ; so that the event loop knows low mem

                lda #$4000              ; and signal the memory manager how
                sta Result,s            ; much mem we freed
;
; Now return to the memory manager first adjusting the stack to remove the
; passed params
OOMEnd
                LongA Off               ; turn on 8 bit accumulator
                SEP #$20

                pla                     ; load the return address for safe
                ply                     ; keeping for a sec

                plx                     ; now pull off 6 bytes of parameters
                plx
                plx

                phy                     ; put the return addr back
                pha
                LongA On                ; turn on 16 bit accumulator
                REP #$20

                RTL                     ; and return
```