Subject: Memory Allocation under GS/OS
Posted by Oz on Tue, 07 Oct 2014 16:39:40 GMT
View Forum Message <> Reply to Message

Games need memory to load their resources (graphic, music, sound...). Most of the Apple IIgs files are 32 KB (graphic page size) or 64 KB (DOC Ram sound file). Because of the Bank boundary, it is easy to manipulate 64 KB (or less) files, and harder to deal with larger file. Even code blocks are limited to 64 KB segments.

When it is time to ask GS/OS some chunk of memory, we could of course ask for the exact size we need (18 KB if the graphic we want to load is 18 KB long) but it is more convenient to ask for whole Bank. So we allocate 64 KB space segments, starting from $XX/0000 to $XX/FFFF, and we organize ourselves the inner storage. The 64 KB segments are perfect to store 2 graphic screens (32 KB each), they can be used for loading area (a 32 KB graphic file, once compressed will be smaller than 32 KB) and the unpack area (we need 32 KB to expand the compressed file).

Allocating one Bank also make the address management more easier because we have only one byte (the bank number, from $00/ to $FF/) to manage and not the 3 bytes address ($E1/2000).

Here is a code to allocate the One Bank from GS/OS :

```
*-------- Allocate 1 bank of 64 KB  --------------------
AllocOneBank   PushLong   #0
               PushLong   #$10000            ; 64 KB
               PushWord   myID               ; ID of my current Program
               PushWord   #%11000000_00011100
               PushLong   #0
               _NewHandle
               PLX                           ; Handle Low Address
               PLA                           ; Handle High Address (00XX)
               XBA
               STA        AOB_1+2
AOB_1          LDAL       $000001,X          ; Get Bank Address (A=XX/00)
               RTS
*-----------------------------------------------------
```

We simply ask the Memory Manager (_NewHandle is a Merlin 16+ Macro to call the Memory Manager) a memory block of 64 KB, aligned on a Bank boundary (so we ensure to get the $XX/0000-$XX/FFFF area).

The Memory Manager do not return a pointer to the area but an Memory Handle (= pointer of pointer). Because we don't want to manipulate the area through it's handle, we need to get the pointer (bank address). So we pull from the Stack the address of the Handle (4 bytes) and we read a 16 bit Word containing the Bank byte ($XX/00) of the allocated Bank. We can perform some self-modification of code (patching the LDAL $000001,X address) because our code is running in RAM. This kind of manipulation is not possible in ROM. If the memory allocation failed, the Carry will be raised.

We can use the AllocOneBank routine at the beginning of the source code, after the Tools initialization :

```
*--------  Memory Allocation  -------------
            JSR       AllocOneBank       ; One Bank for Loading / Unpacking the files
            STA       BankLoadUnpack
*--
            JSR       AllocOneBank       ; One Bank for the Pictures
            STA       BankPicture
*--
            JSR       AllocOneBank       ; One Bank for the Sound
            BCS       Error              ; Test only the latest allocation
            STA       BankSound
...
BankLoadUnpack HEX       0000      ; 00XX
BankPicture    HEX       0000
BankSound      HEX       0000
```

We don't need to care about freeing the memory allocated. The _DisposeAll found at the end of the code will de-allocate everything.

Olivier