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

Subject: Re: Memory Allocation under GS/OS
Posted by AppleIIGSMarc on Fri, 10 Oct 2014 00:32:41 GMT
View Forum Message <> Reply to Message

I don't have easy access to the Toolbox References at the moment (my copies are buried somewhere in a storage unit), so maybe you can help me out with something related. I've started messing around with graphics programming on the IIGS again and so far I've just been writing to the shadow memory in $01/2000 without actually asking for it. Since I'm starting to work on something that I'd actually like to release one day, I need to do it the proper way and request the memory through the memory manager. What parameters can I pass into the NewHandle call to get this specific block of memory and mark it as non-moveable? And yes...I promise to never stomp on memory again without asking first.

Thanks!

Subject: Re: Memory Allocation under GS/OS
Posted by Dagen on Fri, 10 Oct 2014 02:14:00 GMT
View Forum Message <> Reply to Message

You don't need to request/allocate the SHR display area. At least I don't think so. I don't believe that the Memory Manager handles that area. There are some special areas (like Bank 00 as well), which it doesn't touch. I'll try to find the documentation on this, or someone with more knowledge can reply. My understanding was that it's always safe to write to SHR areas at 01/2000-9FFF and E1/2000-9FFF (assuming you're the only program running of course and not an NDA or something).

Of course, before I submit this post, I found a technote (http://apple2.gs/technotes/tn/iigs/TN.IIGS.052.txt) which indicates it's possible to load a program into some of the "Special Memory" such as the SHR area. But I've never had this problem. I only use the Memory Manager to allocate memory for files I load like my graphics and sounds. I just write directly to SHR RAM for graphics. There really is no other way and even if someone else has allocated it that seems more like an indication of a bigger problem that I can't resolve. Just do it. What could possibly go wrong. ;)

Subject: Re: Memory Allocation under GS/OS
Posted by Oz on Fri, 10 Oct 2014 07:00:56 GMT
View Forum Message <> Reply to Message

Hi,

Quote:I've started messing around with graphics programming on the IIGS again and so far I've just been writing to the shadow memory in $01/2000 without actually asking for it. Since I'm starting to work on something that I'd actually like to release one day, I need to do it the proper way and request the memory through the memory manager.

The $01/2000 area may be use for anything else than as a second graphic page area, so it is better to ask for it (in order to prevent any other software to get it). You reserve this area in the same way you allocate other memory parts, using a _NewHandle call :

```
PushLong  #0                ; Ask Shadowing Screen ($8000 bytes from $01/2000)
PushLong  #$8000
PushWord  myID
PushWord  #%11000000_00000011
PushLong  #$012000
_NewHandle
PLA
PLA
```

The difference with a dynamic memory allocation routine as we have seen in the first part of this topic is about the address of the area. We don't need to get the pointer to the area because we already know its address ! We use anyway two PLA to extract the handle from the stack, even if we don't use it.

This code has been already included in the GS/OS programming 'minimum' code provided in

the Sample there : http://apple2.gs/forum/index.php?t=msg&th=15&start=0

Olivier

Subject: Re: Memory Allocation under GS/OS
Posted by AppleIIGSMarc on Fri, 10 Oct 2014 17:14:51 GMT
View Forum Message <> Reply to Message

Thanks, I just went back and looked over that code sample you posted and it's very helpful. Since drive space and memory are no longer much of an issue (thanks to emulators, the CFFA300, and inexpensive memory cards), I'd like everything I work on to be GS/OS compatible and hard drive installable. Popping in a self-booting flopping isn't nearly as much fun as it used to be!

Subject: Re: Memory Allocation under GS/OS
Posted by Oz on Sat, 11 Oct 2014 13:42:52 GMT
View Forum Message <> Reply to Message

Before asking the Memory Manager to allocate us memory space, there is an easy way to know if we can get the expected amount of memory. Because we allocate memory by block of 64 KB (one full Bank), the only thing we care is how many banks the GS/OS can allocate us.

The code is, as usually, very simple :

```
        PushLong  #0              ; Ask Memory Manager for Free Space size
        _FreeMem
        PLA
        PLA
*--
        CMP     #$0007            ; We need 7 Bank of 64 KB
        BCC     ErrorMemory       ; not enough memory, so exit
```

The FreeMem returns the free space size in Bytes, as a 32 bits integer value. We ignore the low WORD and focus on the high one. We have there the number of 64 KB areas available.

We have anyway to check each AllocOneBank calls because of the memory fragmentation (we ask for full banks, not areas that could be located over a bank boundary).

Olivier
