

EN2-Object Heap Discipline

Author: Conrad Kopala

MacApp uses its own object heap to store TObjec derived objects. This object heap grows as new objects are created by taking memory from free memory. Once memory has been allocated to the object heap, it is never ever returned to free memory. I think that causes problems in extreme low memory situations.

The code described in this section erects a barrier beyond which the object heap cannot expand. Best of all, you get to decide where that barrier should be.

To accomplish this, a global variable gOHRemainingIncrements is defined. It keeps a count of how many times the object heap will be allowed to expand. It is initialized to 3 upon startup because MacApp's initialization does two allocations, once to get going and once to set up the basic object heap. It's set to three just to make sure.

Once the application gets going, right after InitUMacApp(moreMasters), the value of gOHRemainingIncrements is reset by the global function InitMaxObjectHeapSize(). Thereafter, whenever more object heap space is requested, gOHRemainingIncrements is tested. If it is greater than zero, the allocation is made and gOHRemainingIncrements is decremented. If it is zero, the allocation is not made and failure is signaled.

The changes to MacApp are confined to PlatformMemory.h and PlatformMemory.cp.

I. PlatformMemory.h additions

Add the global variable

```
extern short gOHRemainingIncrements;
```

II. PlatformMemory.cp additions

Add the global variable and initialize it to 3;

```
short gOHRemainingIncrements = 3;
```

III. PlatformMemory.cp changes

Replace PlatformAllocateBlock() with the following:

```
//-----  
// PlatformAllocateBlock  
//-----  
#pragma segment MAMemoryRes  
  
void *PlatformAllocateBlock(size_t size)  
{  
    Boolean heapPerm;  
  
    if (gUMemoryInitialized)  
        heapPerm = PermAllocation(TRUE);  
  
    void *ptr = NULL;                                     //added by C. Kopala 6/21/96  
    //void *ptr = NewPtr(size);                             //commented out by C. Kopala 6/21/96  
    if (gOHRemainingIncrements > 0)                       //added by by C. Kopala 6/21/96  
    {                                                     //added by by C. Kopala 6/21/96  
        ptr = NewPtr(size);                             //added by by C. Kopala 6/21/96  
        gOHRemainingIncrements--;                       //added by by C. Kopala 6/21/96  
    }                                                     //added by by C. Kopala 6/21/96  
  
    if (gUMemoryInitialized)  
        PermAllocation(heapPerm);                       // Reset perm flag before possible Failure
```

```

        FailNIL(ptr);
    return ptr;
}

```

III. YourApplication.h globals

Define the following constant:

```
const long kFreeMemReserve = 400*1024;
```

Free memory will not be allowed to fall below the value of kFreeMemReserve.

Set it to whatever value is appropriate. Here it is shown set to 400*1024. However, every application will be different and you may have to experiment to determine a proper value.

Add the following global function to your application headers:

```
short InitMaxObjectHeapSize();
```

IV. YourApplication.cp globals

Add the following function to your application:

```

//-----
// InitMaxObjectHeapSize:
//-----
#pragma segment MAGlobalRes
short InitMaxObjectHeapSize()
{
    long freeMem = FreeMem();
    Size heapSizeIncrement = gSizeHeapIncrement;
    short theNumber = 0;

    if (freeMem > kFreeMemReserve)
        theNumber = (freeMem - kFreeMemReserve)/heapSizeIncrement;

    if (theNumber >= 1)
        theNumber = theNumber -1;
    else
        theNumber = 0;

    return theNumber; //The number of times we'll let the object heap be expanded.
}

```

V. MYourApp.cp

In MYourApp.cp, immediately after InitUMacApp set the value of gOHRemainingIncrements by calling InitMaxObjectHeapSize as shown below:

```

void main ()
{
    InitUMacApp(moreMasters);
    gOHRemainingIncrements = InitMaxObjectHeapSize();
}

```