

实验四 内存管理算法改进

实验性质：验证+设计

建议学时：2 学时

任务数：3个

一、 实验目的

- 在 EOS 操作系统中实现边界标识法。
 - 在应用程序中调用 malloc 函数和 free 函数分配和释放内存。

二、实验内容

在 EOS 操作系统内核中，已经通过 mm/mempool.c 文件中的 PoolAllocateMemory 和 PoolFreeMemory 函数实现了动态内存分配，读者可以观察一下 EOS 内核中都有哪些内容是通过动态内存分配来获取内存的，并填写下表。提示：在 EOS 操作系统内核中 PoolAllocateMemory 函数是在 MmAllocateSystemPool 函数中调用的，可以通过查找 MmAllocateSystemPool 函数的调用位置来通过动态内存分配来获取内存的位置。

2.1. 任务（一）：实现控制台命令 dynmm，输出伙伴算法管理的内存数据。

准备实验

请读者按照下面方法之一在本地创建项目，用于完成本次实验任务：

方法一：从 CodeCode.net 平台领取任务

读者需要首先登录 CodeCode.net 平台领取本次实验对应的任务，从而在 CodeCode.net 平台上创建了个人项目，然后使用 OS Lab 提供的“从 Git 远程库新建项目”功能将这个个人项目克隆到本地磁盘中。

方法二：不从 CodeCode.net 平台领取任务

如果读者不使用 CodeCode.net 平台，可以使用 OS Lab 提供的“从 Git 远程库新建项目”功能直接将 EOS 内核实验模板克隆到本地磁盘中，创建一个 EOS 内核项目，实验模板的

URL 为

<https://www.codecode.net/engintime/os-lab/Project-Template/eos-kernel.git>

完成实验

EOS 操作系统内核使用伙伴算法对动态分配的内存进行管理，EOS 操作系统内核与实验 9 中实现的伙伴算法函数的对应关系如下表，读者可以仿照实验 9 提供的模式，在 EOS 内核中添加一个控制台命令“dynmm”，将伙伴算法管理的内存数据（包括已使用内存块的数量与已使用内存块总的大小，空闲的内存块数量与空闲内存块总的大小）打印输出到屏幕上。

EOS 操作系统的伙伴算法	实验 9 的伙伴算法
PoolAllocateMemory	AllocBuddy
PoolFreeMemory	Reclaim
PoolInitialize	main 函数的前半部分
struct _MEM_BLOCK	struct _Node

```
>dynmm  
freeMemBlockCount = 12, memorySize = 3085952  
usedMemBlockCount = 115, memorySize = 23648
```

提交作业

如果读者是通过从 CodeCode.net 平台领取任务创建的 EOS 内核项目，并将项目克隆到本地进行实验，实验结束后可以将本地已更改的项目再推送到 CodeCode.net 平台的个人项目中，方便教师通过 CodeCode.net 平台查看读者提交的作业。

2.2. 任务（二）：将 EOS 内核的伙伴算法替换为边界标识法。

准备实验

请读者按照下面方法之一在本地创建项目，用于完成本次实验任务：

方法一：从 CodeCode.net 平台领取任务

读者需要首先登录 CodeCode.net 平台领取本次实验对应的任务，从而在 CodeCode.net 平台上创建了个人项目，然后使用 OS Lab 提供的“从 Git 远程库新建项目”功能将这个个人项目克隆到本地磁盘中。

方法二：不从 CodeCode.net 平台领取任务

如果读者不使用 CodeCode.net 平台，可以使用 OS Lab 提供的“从 Git 远程库新建项目”功能直接将 EOS 内核实验模板克隆到本地磁盘中，创建一个 EOS 内核项目，实验模板的 URL 为

<https://www.codecode.net/engintime/os-lab/Project-Template/eos-kernel.git>

实现边界标识法的内存分配算法

待读者对 EOS 操作系统内核中的动态内存分配机制有一个深入的了解后，可以尝试将 EOS 操作系统中的伙伴算法修改为边界标识法，EOS 操作系统内核实现的伙伴算法与实验 9 中实现的边界标识法的函数对应关系如下表，读者只需要替换对应函数中的代码，并对代码做适当的调整即可在 EOS 操作系统内核中实现边界标识法。这里给出一些提示信息：

EOS 操作系统的伙伴算法	实验 9 的边界标识法
PoolAllocateMemory	allocBoundTag
PoolFreeMemory	reclaimBoundTag
PoolInitialize	initSpace
struct _MEM_BLOCK	struct Bound

- 可以先实现边界标识法中的内存初始化和内存分配算法，回收算法可以暂时不实现。
- 在实现边界标识法的内存分配算法时，需要注意，实际分配的内存大小为：定义的结构体的大小与需要分配的内存大小之和，返回的地址为：内存块的起始地址与定义的结构体的大小之和，如果直接返回内存块的起始地址，在写内存时会将结构体信息覆盖，破坏内存块的结构数据。
- 在实现分配算法后，尝试启动内核，如果正常启动，可以输入 pt 命令和执行应用程序 Hello.exe 进行测试，如果可以正常执行，说明已实现的内存分配算法可以正常使用。

~~实现控制台命令“dynmm”的相关代码，将边界标识法管理的内存数据打印输出到屏幕上。~~

~~通过实现控制台命令“dynmm”的相关代码，将边界标识法管理的内存数据打印输出到屏幕上。输出的内容可以参考任务。~~

~~实现边界标识法的内存回收算法。~~

~~这里给出一些提示信息：~~

- 在 EOS 内核中，实现边界标识法的内存回收算法。
- 实现后，尝试启动内核，确保可以正常启动，并且可以执行应用程序和控制台命令。
- 在控制台中输入 dynmm 命令，查看空闲块的数量和已使用块的数量，在回收内存后，已使用块的数量和任务二相比已经减少。

提交作业

如果读者是通过从 CodeCode.net 平台领取任务创建的 EOS 内核项目，并将项目克隆到本地进行实验，实验结束后可以将本地已更改的项目再推送到 CodeCode.net 平台的个人项目中，方便教师通过 CodeCode.net 平台查看读者提交的作业。

2.3. 任务（三）：在 EOS 应用程序中实现 malloc、calloc、realloc 和 free 函数

准备实验

请读者按照下面方法之一在本地创建项目，用于完成本次实验任务：

方法一：从 CodeCode.net 平台领取任务

读者需要首先登录 CodeCode.net 平台领取本次实验对应的任务，从而在 CodeCode.net 平台上创建了个人项目，然后使用 OS Lab 提供的“从 Git 远程库新建项目”功能将这个个人项目克隆到本地磁盘中。

方法二：不从 CodeCode.net 平台领取任务

如果读者不使用 CodeCode.net 平台，可以使用 OS Lab 提供的“从 Git 远程库新建项目”功能直接将 EOS 内核实验模板克隆到本地磁盘中，创建一个 EOS 内核项目，实验模板的 URL 为

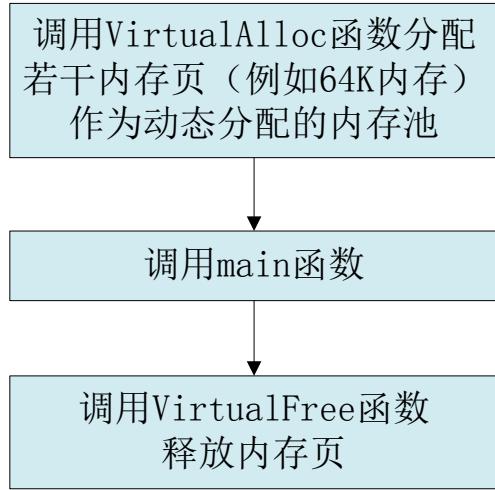
<https://www.codecode.net/engintime/os-lab/Project-Template/eos-app.git>

完成实验

在 EOS 应用程序提供的 C 运行时库中，还没有实现动态内存分配，也就是说无法在 EOS 应用程序的 C 源代码中调用 malloc、calloc、realloc 和 free 函数。读者可以尝试在 EOS 应用程序的 crt/src/stdlib.c 文件中实现与 C 运行时标准库一致的 malloc、calloc、realloc 和 free 函数。这里给出一些提示信息：

- 首先需要修改 EOS 应用程序 crt/src/crt0.c 文件中的标准库初始化函数_start，在其中调用 EOS API 函数 VirtualAlloc 分配若干个内存页作为动态分配的内存池，并将其基址映射到进程用户空间（低 2G）中的合适位置。注意分配内存页的代码

要在调用 main 函数执行用户编写的功能之前，之后还需要调用 VirtualFree 函数释放内存页。可以参考下面的流程图。



- 在 crt/src/stdlib.c 文件中定义必要的数据结构，实现 malloc、calloc、realloc 和 free 函数。可以使用边界标识法，也可以使用伙伴算法等。
- 在 EOS 应用程序的 main 函数中尝试调用 malloc、calloc、realloc 函数分配内存，然后将两个整数写入已分配的内存中，求两个整数的和，并将求得的结果写入到已分配的内存中，在对内存进行读写访问后调用 free 函数释放内存，确保动态内存分配功能可以正常工作。在 EOS 应用程序的 main 函数中调用 malloc 和 free 函数并读写内存的测试代码，如下表，并以伙伴算法为例输出应用程序的内存使用数据。

```
int main(int argc, char* argv[])
{
    printf("=====malloc memory=====\\n");
    PINT pValue = (PINT)malloc(20);
    if(NULL == pValue) {
        printf("PValue Allocate virtual memory failed!\\n\\n");
        return;
    }

    PINT pValue2 = (PINT)malloc(25);
    if(NULL == pValue2) {
        printf("PValue2 Allocate virtual memory failed!\\n\\n");
        return;
    }

    PINT pValue3 = (PINT)malloc(sizeof(INT));
    if(NULL == pValue3) {
        printf("PValue2 Allocate virtual memory failed!\\n\\n");
        return;
    }
    *pValue3 = 5;
```

```

PINT pValue4 = (PINT)malloc(sizeof(INT));
if(NULL == pValue2) {
    printf("PValue2 Allocate virtual memory failed!\n\n");
    return;
}

*pValue4 = 8;
*pValue4 += *pValue3;
printf("%d + %d = %d\n", *pValue3, *pValue4, *pValue3 + *pValue4);

print_used(); // 输出内存的使用情况

printf("=====free memory=====\n");
free((VOID)pValue);

print_used();

return 0;
}

```

```

Welcome to EOS shell
>Autorun A:\eosapp.exe
=====
UsedSpace:
UsedBlockNum      UsedBlockBeginAddr      BlockSize      BlockTag(0:Free 1:Used)
0                  0x10000                 32             1
1                  0x10020                 32             1
2                  0x10040                 8192           1
3                  0x10048                 8              1

=====
UsedSpace:
UsedBlockNum      UsedBlockBeginAddr      BlockSize      BlockTag(0:Free 1:Used)
1                  0x10020                 32             1
2                  0x10040                 8192           1
3                  0x10048                 8              1

```

- 还可以进一步修改 EOS 应用程序 crt/src/crt0.c 文件中的标准库初始化函数 _start，在其中调用 EOS API 函数 VirtualAlloc 动态分配若干个内存页作为动态分配的内存池，如果动态分配的内存池内存不足之后，还可以继续调用 VirtualAlloc 动态分配若干个内存页作为动态分配的内存池，直到操作系统没有可以分配的内存页。

读者完成以上练习后，可以思考以下问题：

- 如果 EOS 应用程序只分配了内存，而未释放内存，在应用程序退出后，EOS 操作系统会回收这部分内存吗？
- 应用程序进程异常结束，杀死应用程序进程后，EOS 操作系统如何回收内存？如果 EOS 操作系统不回收这部分内存会有什么后果？
- VirtualAlloc 是在低 2G 还是在高 2G 的虚拟地址空间分配的内存？VirtualAlloc 如果在低 2G 虚拟地址空间分配的内存，可以尝试在 malloc 中直接调用 VirtualAlloc 分配内存，在 free 中调用 VirtualFree 函数释放内存。

- 由于在 `malloc` 中直接调用 `VirtualAlloc` 分配内存，会使在应用程序中每次调用 `malloc` 时，都会进入内核，不利于应用程序自己管理内存空间，并会导致执行效率低下。如果直接在 `malloc` 函数中实现动态内存分配算法（边界标识法或伙伴算法），每次调用 `malloc` 都会从应用层调用，执行效率会高很多。通过分别执行程序比较调用以上两种方法的执行效率。

提示：如果需要将地址和整数进行相加得到偏移地址，需要先将地址转换为无符号长整型（`ULONG`）后，再进行运算。

提交作业

如果读者是通过从 CodeCode.net 平台领取任务创建的 EOS 应用程序项目，并将项目克隆到本地进行实验，实验结束后可以将本地已更改的项目再推送到 CodeCode.net 平台的个人项目中，方便教师通过 CodeCode.net 平台查看读者提交的作业。