

北京科技大学实验报告

学院：计算机与通信工程学院 专业：计算机科学与技术 班级：计 1703

姓名：张宝丰

学号：41724081

实验日期：2019 年 12 月 1 日

实验名称：操作系统实验 5 物理存储器与进程逻辑地址空间管理（2 分）

实验目的：以一个教学型操作系统 EOS 为例，深入理解物理存储器以及进程逻辑地址空间的管理方法；能对核心源代码进行分析；训练分析问题、解决问题以及自主学习能力，逐步达到能独立对小型操作系统的功能进行分析、设计和实现。

实验环境：EOS 操作系统及其实验环境。

实验内容：

通过查看 EOS 物理存储器的使用情况，练习物理内存的分配与回收，分析相关源代码；通过查看进程逻辑地址空间的使用情况，练习虚拟内存的分配与回收，分析相关源代码，完成在应用进程中分配虚拟页和释放虚拟页的功能。

实验步骤：

1) EOS 物理内存分配和回收的练习以及源代码分析

（练习物理内存的分配和回收；分析相关源代码，阐述 EOS 中物理存储器的管理方法，包括数据结构和算法等；简要说明在本部分实验过程中完成的主要工作）

1. EOS 物理存储器的管理方法

EOS 使用分页内存管理方式，用页框号数据库 PFN Database 来管理所有物理页，PFN 其实是一个数组，其中第 N 项描述了页框号为 N 的物理页的状态，并且结构体中的第三项 Next 指向了下一个物理页的页框号，页框结构体的定义如下：

```
typedef struct _MMPFN
{
    ULONG Unused : 9;    // 未用
    ULONG PageState : 3; // 页的状态
    ULONG Next : 20;
}MMPFN, *PMMPFN;
```

EOS 目前只定义了三种物理页的页状态，包括零页、自由页、占用页，页的状态被定义为枚举类型，在文件 mm/mi.h 定义如下：

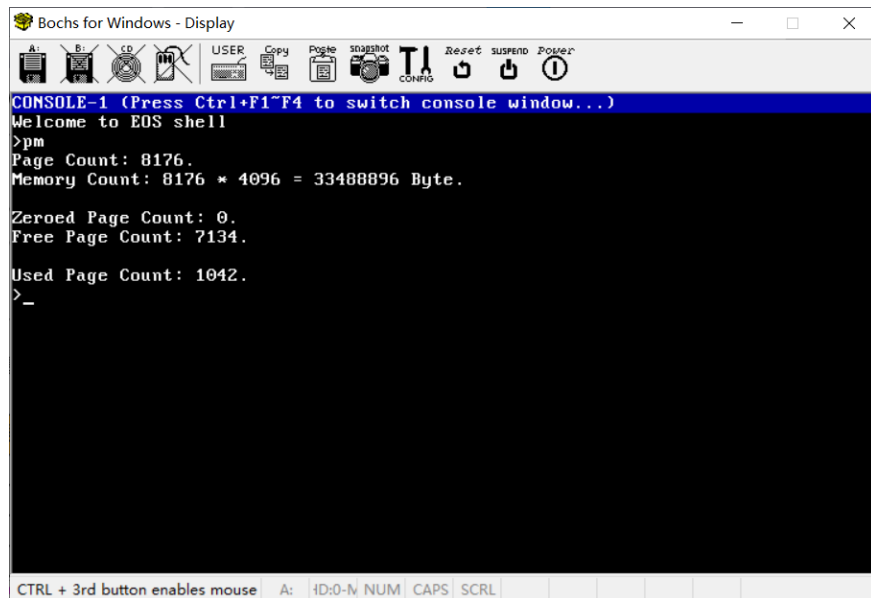
```
typedef enum _PAGE_STATE {
    ZEROED_PAGE, // 0
    FREE_PAGE,   // 1
    BUSY_PAGE,   // 2
}
```

```
} PAGE_STATE;
```

在开启 i386 的分页功能后，也就是核心 kernel.dll 被加载进入内存并运行之后，就不能通过物理地址访问内存了，任何操作都必须通过逻辑地址再映射到物理地址，因此需要事先准备好一些物理页用于存放目录和页表。EOS 在开启分页功能时，就将页目录和所有页表映射在虚拟地址 0xC0000000 开始的 4MB 地址空间中，以满足修改页目录和页表的需求。

2. 练习物理内存的分配和回收

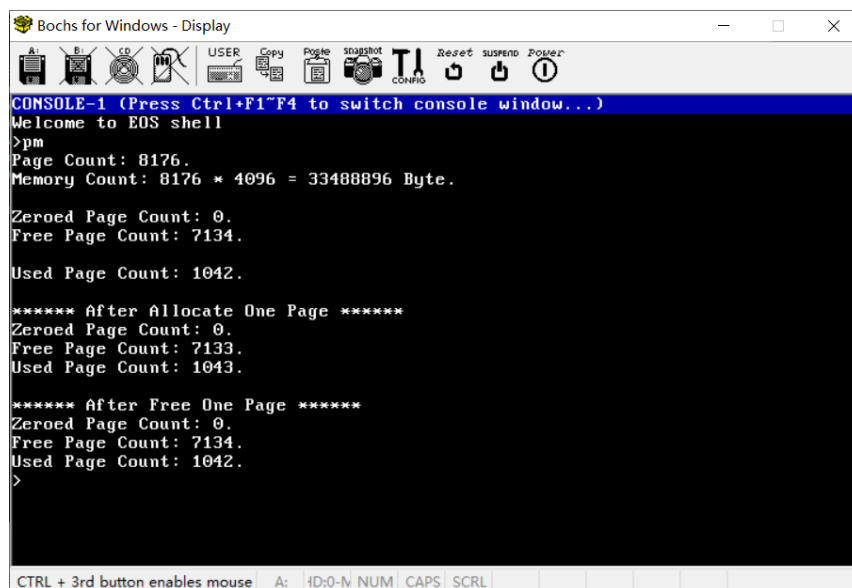
控制台的 pm 指令可以查看物理存储器的信息，下图是修改前 pm 的运行结果：



```
Bochs for Windows - Display
A: B: CD USER Copy Paste snapshot T1 Reset suspend Power
CONFIG
CONSOLE-1 (Press Ctrl+F1~F4 to switch console window...)
Welcome to EOS shell
>pm
Page Count: 8176.
Memory Count: 8176 * 4096 = 33488896 Byte.

Zeroed Page Count: 0.
Free Page Count: 7134.
Used Page Count: 1042.
>_
CTRL + 3rd button enables mouse  A: ID:0-N NUM CAPS SCRL
```

在修改 ConsoleCmdPhysicalMemory 函数之后，函数分配一个物理页再将其释放，下面是它的运行结果：



```
Bochs for Windows - Display
A: B: CD USER Copy Paste snapshot T1 Reset suspend Power
CONFIG
CONSOLE-1 (Press Ctrl+F1~F4 to switch console window...)
Welcome to EOS shell
>pm
Page Count: 8176.
Memory Count: 8176 * 4096 = 33488896 Byte.

Zeroed Page Count: 0.
Free Page Count: 7134.
Used Page Count: 1042.

***** After Allocate One Page *****
Zeroed Page Count: 0.
Free Page Count: 7133.
Used Page Count: 1043.

***** After Free One Page *****
Zeroed Page Count: 0.
Free Page Count: 7134.
Used Page Count: 1042.
>
CTRL + 3rd button enables mouse  A: ID:0-N NUM CAPS SCRL
```

MiAllocate 问题解答:

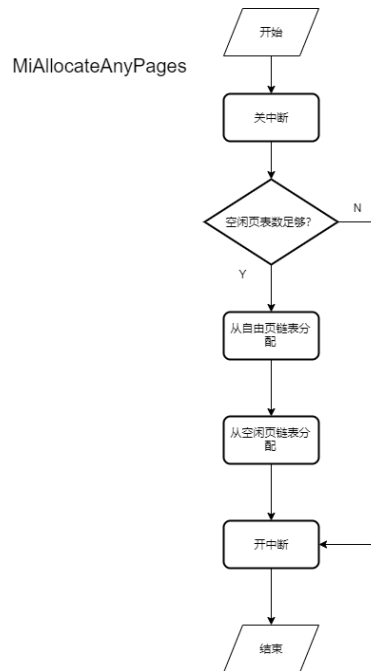
(1) 分配 1 个物理页, 分配的物理页的页框号是 0x409;

(2) 从空闲链表 (MiFreePageList) 分配;

(3) MiFreePageCount--; // 减少空闲页的数量;

MiGetPfnDatabaseEntry(Pfn)->PageState = BUSY_PAGE; // 修改页的状态为忙状态

(4) MiAllocate 流程图:

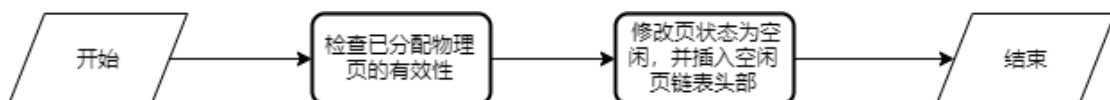


MiFreePages 问答:

(1) 释放 1 个物理页, 释放物理页的页框号是 0x409, 正是之前分配的页框号;

(2) 释放的物理页放入了空闲页链表中;

(3) MiFreePages 流程图:



2) EOS 进程逻辑地址空间分配和回收的练习以及源代码分析

(练习虚拟内存的分配和回收; 分析相关源代码, 阐述 EOS 中进程逻辑地址空间的管理方法, 包括数据结构和算法等; 给出在应用进程中分配虚拟页和释放虚拟页的实现方法简要描述、源代码、测试及结果等; 简要说明在本部分实验过程中完成的主要工作)

1. EOS 进程逻辑地址空间的管理方法

EOS 使用一个虚拟地址描述符 (VAD) 来记录一段被使用的地址范围, 并将所有 VAD 按照地址增序组成链表, 其定义如下:

// VAD 结构体

```
typedef struct _MMVAD{
```

```
    ULONG_PTR StartingVpn;    // 被使用区域的开始虚页框号
```

```
    ULONG_PTR EndVpn;        // 被使用区域的结束虚页框号
```

```

LIST_ENTRY VadListEntry;    // 链表项
}MMVAD, *PMMVAD;

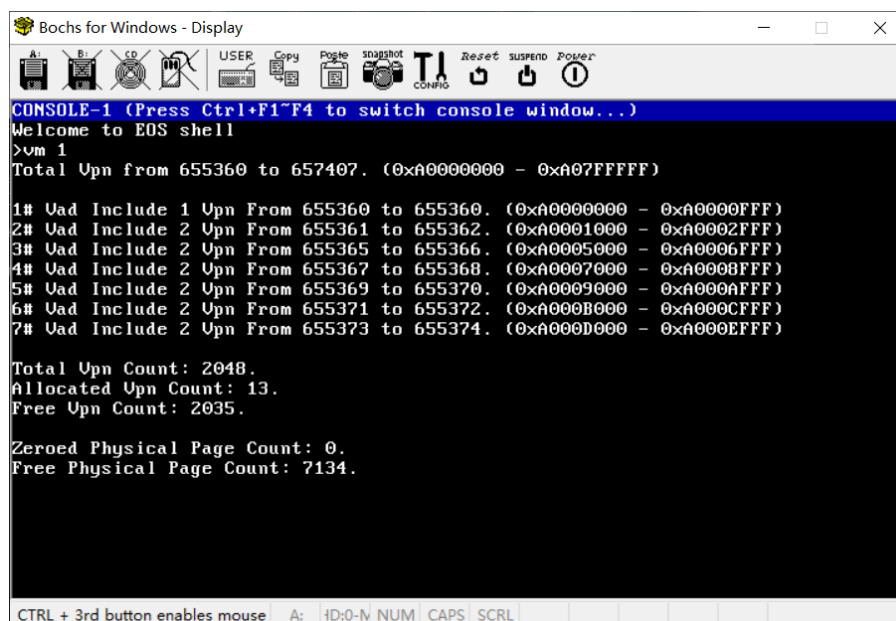
// VAD 链表结构体
typedef struct _MMVAD_LIST{
    ULONG_PTR StartingVpn;    // 记录的进程地址空间的开始虚页号
    ULONG_PTR EndVpn;        // 记录的进程地址空间的结束虚页号
    LIST_ENTRY VadListHead;   // VAD 链表头
}MMVAD_LIST, *PMMVAD_LIST;

```

用户进程可以调用 VirtualAlloc 来分配虚拟内存，VirtualAlloc 会遍历 VAD 链表，查找是否有符合用户要求的起始地址和长度的一段内存空间，并进行分配。释放虚拟内存的函数是 VirtualFree，算法也会同时将虚拟地址映射的物理地址一并释放。

2. 虚拟内存的分配和回收过程

vm 1 的虚拟地址描述符



分配部分的问答：

- (1) 分配虚拟页的起始地址是 0xa003000，数量是 1；

```

//
// 记录实际保留的起始、结束虚页框号。
//
StartingVpn = Vad->StartingVpn;
EndVpn = StartingVpn = 0xa003000;

```

(2) 分配虚拟页的同时没有映射实际的物理页，函数只选择了 MEM_RESERVE，而没有选择 MEM_COMMIT；

- (3) 分配的地址空间在系统地址空间（高 2G），由 SystemVirtual=0x1 指定。

- (4) MiReserveAddressRegion 在地址空间中保留一段地址区域，保留的单位为页，该

函数为 COMMIT 操作预留空间；若空间未被使用，再调用 MiFreeAddressRegion 将其释放。

3. 在应用进程中分配虚拟页和释放虚拟页的实现方法

```
#include "EOSApp.h"
```

```
int main(int argc, char* argv[])
```

```
{
```

```
    int *d;
```

```
    // 为变量分配内存空间
```

```
    if(d = VirtualAlloc(0,sizeof(int),MEM_RESERVE|MEM_COMMIT)){
```

```
        printf("Allocate %d bytes virtual memory at 0x%x\n\n",sizeof(int),*d);
```

```
        printf("Virtual memory original value:0x%x\n\n",*d);
```

```
        // 修改变量的值
```

```
        *d = 0xFFFFFFFF;
```

```
        printf("Virtual memory new value:0x%x\n\n",*d);
```

```
        // 等待 10s
```

```
        Sleep(10000);
```

```
        printf("Wait for 10 second\n\n");
```

```
    }
```

```
    if(VirtualFree(d,0,MEM_RELEASE)==1)
```

```
        // 释放成功，打印信息
```

```
        printf("SUCCESS\n");
```

```
    else
```

```
        // 释放失败
```

```
        printf("ERROR\n");
```

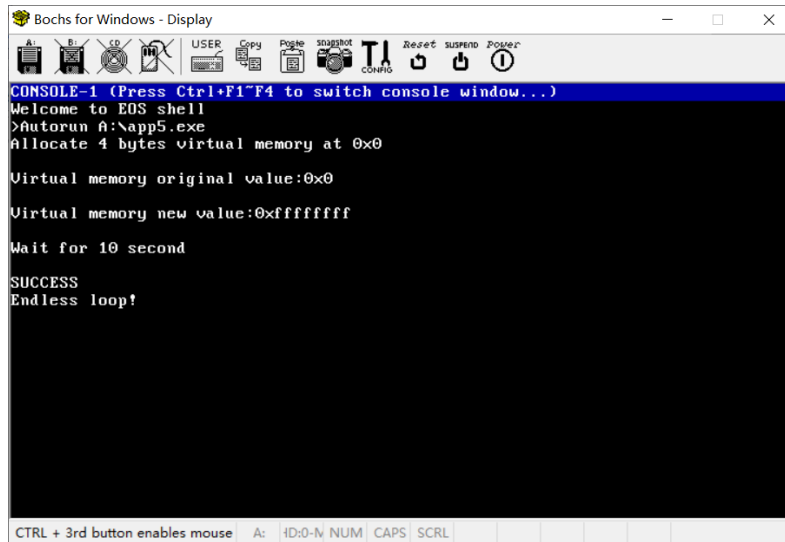
```
    printf("Endless loop!\n");
```

```
    for(;;){}
```

```
    return 0;
```

```
}
```

验证结果：



结果分析:

(对本实验所做工作及结果进行分析,包括 EOS 物理存储器管理与进程逻辑地址空间管理方法的特点、不足及改进意见;结合 EOS 对物理存储器与进程逻辑地址空间管理相关问题提出自己的思考;其他需要说明的问题)

EOS 使用分页内存管理方式,用页框号数据库 PFN Database 来管理所有物理页;EOS 使用一个虚拟地址描述符(VAD)来记录一段被使用的地址范围,并将所有 VAD 按照地址增序组成链表,以此来管理虚拟页。EOS 指定低 2G 为用户地址空间,高 2G 为系统地址空间。