同济大学计算机系

操作系统实验报告

学　　号 2152809

姓　　名 曾崇然

专　　业 计算机科学与技术

授课老师 邓蓉老师

完成内容：完成了实验步骤 1 和 2，即读通了程序 bing 完成了将相对虚实地址映射表赋值为 NULL 并跑通

一. 读子程序并进行注释

1. Initialize:

初始化 MemoryDescriptor

```
//初始化MemoryDescr
void MemoryDescriptor::Initialize()
{
    KernelPageManager& kernelPageManager = Kernel::Instance().GetKernelPageMana

    /* m_UserPageTableArray需要把AllocMemory()返回的物理内存地址 + 0xC0000000 */
    //分配两个页框装相对地址映射表并计算其虚拟地址
    this->m_UserPageTableArray = (PageTable*)(kernelPageManager.AllocMemory(size
}
```

2. Release:

释放 MemoryDescriptor 对象

```
//释放MemoryDescriptor对象
void MemoryDescriptor::Release()
{
    KernelPageManager& kernelPageManager = Kernel::Instance().GetKernelPageManager();

    //释放行对地址映射表占据的两个页框并将相对地址映射表指针置为NULL
    if ( this->m_UserPageTableArray )
    {
        kernelPageManager.FreeMemory(sizeof(PageTable) * USER_SPACE_PAGE_TABLE_CNT, (un
        this->m_UserPageTableArray = NULL;
    }
}
```

3. MapEntry:

写页表

```
//写页表
unsigned int MemoryDescriptor::MapEntry(unsigned long virtualAddress, unsigned int size,
{
    //计算物理地址
    unsigned long address = virtualAddress - USER_SPACE_START_ADDRESS;

    //计算从pagetable的哪一个地址开始映射
    unsigned long startIdx = address >> 12;
    //计算要写多少个PTE
    unsigned long cnt = ( size + (PageManager::PAGE_SIZE - 1) )/ PageManager::PAGE_SIZE;

    //写页表
    PageTableEntry* entrys = (PageTableEntry*)this->m_UserPageTableArray;
    for ( unsigned int i = startIdx; i < startIdx + cnt; i++, phyPageIdx++ )
    {
        entrys[i].m_Present = 0x1;
        entrys[i].m_ReadWriter = isReadWrite;
        entrys[i].m_PageBaseAddress = phyPageIdx;
    }
    return phyPageIdx;
}
```

4. GetUserPageTableArray:

获取相对地址映射表的虚拟地址

```
//获取相对地址映射表的虚拟地址
PageTable* MemoryDescriptor::GetUserPageTableArray()
{
    return this->m_UserPageTableArray;
}
```

5. ClearUserPageTable:

清空相对地址映射表

```
//清空相对地址映射表
void MemoryDescriptor::ClearUserPageTable()
{
    User& u = Kernel::Instance().GetUser();
    PageTable* pUserPageTable = u.u_MemoryDescriptor.m_UserPageTableArray;

    unsigned int i ;
    unsigned int j ;

    for (i = 0; i < Machine::USER_PAGE_TABLE_CNT; i++)
    {
        for (j = 0; j < PageTable::ENTRY_CNT_PER_PAGETABLE; j++ )
        {
            pUserPageTable[i].m_Entrys[j].m_Present = 0;
            pUserPageTable[i].m_Entrys[j].m_ReadWriter = 0;
            pUserPageTable[i].m_Entrys[j].m_UserSupervisor = 1;
            pUserPageTable[i].m_Entrys[j].m_PageBaseAddress = 0;
        }
    }

}
```

6. EstablishUserPageTable:

建立相对地址映射表

```
//建立相对地址映射表
bool MemoryDescriptor::EstablishUserPageTable( unsigned long textVirtualAddress, unsigned long textSize, unsi
{
    User& u = Kernel::Instance().GetUser();

    /* 如果超出允许的用户程序最大8M的地址空间限制 */
    if ( textSize + dataSize + stackSize  + PageManager::PAGE_SIZE > USER_SPACE_SIZE - textVirtualAddress)
    {
        u.u_error = User::ENOMEM;
        Diagnose::Write("u.u_error = %d\n",u.u_error);
        return false;
    }

    this->ClearUserPageTable();

    /* 以相对起始地址phyPageIndex为0，为正文段建立相对地址映照表 */
    unsigned int phyPageIndex = 0;
    phyPageIndex = this->MapEntry(textVirtualAddress, textSize, phyPageIndex, false);

    /* 以相对起始地址phyPageIndex为1，ppda区占用1页4K大小物理内存，为数据段建立相对地址映照表 */
    phyPageIndex = 1;
    phyPageIndex = this->MapEntry(dataVirtualAddress, dataSize, phyPageIndex, true);

    /* 紧跟着数据段之后，为堆栈段建立相对地址映照表 */
    unsigned long stackStartAddress = (USER_SPACE_START_ADDRESS + USER_SPACE_SIZE - stackSize) & 0xFFFFF000;
    this->MapEntry(stackStartAddress, stackSize, phyPageIndex, true);

    /* 将相对地址映照表根据正文段和数据段在内存中的起始地址pText->x_caddr、p_addr，建立用户态内存区的页表映射 */
    this->MapToPageTable();
    return true;
}
```

7. MapToPageTable:

根据相对地址映射表刷新页表

```
//根据相对地址映射表刷新页表
void MemoryDescriptor::MapToPageTable()
{
    User& u = Kernel::Instance().GetUser();
    PageTable* pUserPageTable = Machine::Instance().GetUserPageTableArray();
    unsigned int textAddress = 0;
    if ( u.u_procp->p_textp != NULL )
    {
        textAddress = u.u_procp->p_textp->x_caddr;
    }

    for (unsigned int i = 0; i < Machine::USER_PAGE_TABLE_CNT; i++)
    {
        for ( unsigned int j = 0; j < PageTable::ENTRY_CNT_PER_PAGETABLE; j++ )
        {
            pUserPageTable[i].m_Entrys[j].m_Present = 0;    //先清0

            if ( 1 == this->m_UserPageTableArray[i].m_Entrys[j].m_Present )
            {
                /* 只读属性表示正文段对应的页，以pText->x_caddr为内存起始地址 */
                if ( 0 == this->m_UserPageTableArray[i].m_Entrys[j].m_ReadWriter )
                {
                    pUserPageTable[i].m_Entrys[j].m_Present = 1;
```

二. 去除相对地址映射表的重要性

1. 这个数据结构是没有其必要性的：相对虚实地址映射表能够用来写系统页表，但是通过各段的长度和 p_addr，p_text 等信息就能够直接写出系统页表，因此相对虚实地址映射表是冗余的

2. 占用太多核心空间：一个进程需要一张相对虚实地址映射表，一张相对虚实地址映射表占用两个页框大小的核心空间，这徒劳的增加了系统核心空间的开销

综上，去除相对虚实地址映射表不仅不会影响系统的功能，反而能使系统更加的合理和高效。

三. 将相对地址映射表赋值为 NULL 并跑通程序

1. initialize:

将相对虚实地址映射表赋值为 NULL

```
//初始化MemoryDescriptor对象
void MemoryDescriptor::Initialize()
{
    KernelPageManager& kernelPageManager = Kernel::Instance().GetKernelPa

    /****************以下是将相对地址映射表赋值NULL的代码*****************/
    this->m_UserPageTableArray = NULL;

    /****************以下是保留相对地址映射表的代码**********************/
    /* m_UserPageTableArray需要把AllocMemory()返回的物理内存地址 + 0xC0000000 */
    //分配两个页框装相对地址映射表并计算其虚拟地址
    //this->m_UserPageTableArray = (PageTable*)(kernelPageManager.AllocMe
}
```

2. Release:

因为相对虚实地址映射表已经为 NULL 了，所以该函数不做操作，无需修改

3. MapEntry:

在建立相对虚实地址表的函数中调用，现在不使用相对地址映射表，所以将其内容注释掉即可

```
unsigned int MemoryDescriptor::MapEntry(unsigned long virtualAddress, unsigned int si
{

    /****************以下是将相对地址映射表赋值NULL的代码*****************/
    return phyPageIdx;

    /****************以下是保留相对地址映射表的代码**********************/
//  //计算物理地址
//  unsigned long address = virtualAddress - USER_SPACE_START_ADDRESS;
//
//  //计算从pagetable的哪一个地址开始映射
//  unsigned long startIdx = address >> 12;
//  //计算要写多少个PTE
//  unsigned long cnt = ( size + (PageManager::PAGE_SIZE - 1) )/ PageManager::PAGE_SI
//
//  //写页表
//  PageTableEntry* entrys = (PageTableEntry*)this->m_UserPageTableArray;
//  for ( unsigned int i = startIdx; i < startIdx + cnt; i++, phyPageIdx++ )
//  {
```

4. GetUserPageTableArray:

现在相对地址映射表为 NULL，所以直接返回 NULL 即可

```
//获取相对地址映射表的虚拟地址
PageTable* MemoryDescriptor::GetUserPageTableArray()
{
    /*****************以下是将相对地址映射表赋值NULL的代码******************/
    return NULL;

    /*****************以下是保留相对地址映射表的代码**********************/
//  return this->m_UserPageTableArray;
}
```

5. ClearUserPageTable:

因为拿掉了相对虚实地址映射表，所以清除时无需操作，全部注释掉即可

```
//清空相对地址映射表
void MemoryDescriptor::ClearUserPageTable()
{
    /*****************以下是将相对地址映射表赋值NULL的代码*****************/

    /*****************以下是保留相对地址映射表的代码**********************/
//  User& u = Kernel::Instance().GetUser();
//  PageTable* pUserPageTable = u.u_MemoryDescriptor.m_UserPageTableArray;
//
//  unsigned int i ;
//  unsigned int j ;
//
//  for (i = 0; i < Machine::USER_PAGE_TABLE_CNT; i++)
//  {
//      for (j = 0; j < PageTable::ENTRY_CNT_PER_PAGETABLE; j++ )
//      {
//          pUserPageTable[i].m_Entrys[j].m_Present = 0;
//          pUserPageTable[i].m_Entrys[j].m_ReadWriter = 0;
//          pUserPageTable[i].m_Entrys[j].m_UserSupervisor = 1;
//          pUserPageTable[i].m_Entrys[j].m_PageBaseAddress = 0;
//      }
//  }

}
```

6. EstablishUserPageTable:

大小超过 8M 地址空间即报错，小于则为代码段、数据段和堆栈段赋值，并调用 MapToPageTable 刷新页表

```
//建立相对地址映射表
bool MemoryDescriptor::EstablishUserPageTable( unsigned long textVirtualAddress, un
{
    /*****************以下是将相对地址映射表赋值NULL的代码*****************/
    User& u = Kernel::Instance().GetUser();
    if ( textSize + dataSize + stackSize  + PageManager::PAGE_SIZE > USER_SPACE_SIZ
    {
        u.u_error = User::ENOMEM;
        Diagnose::Write("u.u_error = %d\n",u.u_error);
        return false;
    }
    m_TextSize = textSize;
    m_DataSize = dataSize;
    m_StackSize = stackSize;
    this->MapToPageTable();
    return true;

    /*****************以下是保留相对地址映射表的代码**********************/
//  User& u = Kernel::Instance().GetUser();
//
```

7. MapToPageTable:

①计算各段起始偏移量和长度

②遍历每张用户页表的每一项进行写入

③根据代码段、数据段、堆栈的不同特性写其 p 值和 w/r 值，根据 p_text，p_addr 来计算 base 以回避掉相对虚实地址映射表

```
//根据相对地址映射表刷新页表
void MemoryDescriptor::MapToPageTable()
{
    /*****************以下是将相对地址映射表赋值NULL的代码*****************/
    User& u = Kernel::Instance().GetUser();
    PageTable* pUserPageTable = Machine::Instance().GetUserPageTableArray();
    unsigned int textAddress = 0;
    if (u.u_procp->p_textp != NULL) {
        textAddress = u.u_procp->p_textp->x_caddr;
    }
    unsigned int tstart_index = 0, dstart_index = 1;
    unsigned int text_len = (m_TextSize + (PageManager::PAGE_SIZE - 1)) / PageManager::PAGE_SIZE;
    unsigned int data_len = (m_DataSize + (PageManager::PAGE_SIZE - 1)) / PageManager::PAGE_SIZE;
    unsigned int stack_len = (m_StackSize + (PageManager::PAGE_SIZE - 1)) / PageManager::PAGE_SIZE;
    unsigned int dataidx = 0;
    for (unsigned int i = 0; i < Machine::USER_PAGE_TABLE_CNT; i++) {
        for (unsigned int j = 0; j < PageTable::ENTRY_CNT_PER_PAGETABLE; j++) {
            pUserPageTable[i].m_Entrys[j].m_Present = 0;
            if (1 == i) {
                if (1 <= j && j <= text_len) {
                    pUserPageTable[i].m_Entrys[j].m_Present = 1;
                    pUserPageTable[i].m_Entrys[j].m_ReadWriter = 0;
                    pUserPageTable[i].m_Entrys[j].m_PageBaseAddress = j - 1 + tstart_index + (textAddress >> 12);
                }
                else if (j > text_len && j <= text_len + data_len) {
                    pUserPageTable[i].m_Entrys[j].m_Present = 1;
                    pUserPageTable[i].m_Entrys[j].m_ReadWriter = 1;
                    pUserPageTable[i].m_Entrys[j].m_PageBaseAddress = dataidx + dstart_index + (u.u_procp->p_addr >> 12);
                    dataidx++;
                }
                else if (j >= PageTable::ENTRY_CNT_PER_PAGETABLE - stack_len) {
                    pUserPageTable[i].m_Entrys[j].m_Present = 1;
                    pUserPageTable[i].m_Entrys[j].m_ReadWriter = 1;
                    pUserPageTable[i].m_Entrys[j].m_PageBaseAddress = dataidx + dstart_index + (u.u_procp->p_addr >> 12);
                    dataidx++;
                }
            }
        }
    }
    pUserPageTable[0].m_Entrys[0].m_Present = 1;
    pUserPageTable[0].m_Entrys[0].m_ReadWriter = 1;
    pUserPageTable[0].m_Entrys[0].m_PageBaseAddress = 0;
    FlushPageDirectory();
```

四. 实验现象

1. 编译成功

```
            if (1 == i) {
                if (1 <= j && j <= text_len) {
                    pUserPageTable[i].m_Entrys[j].m_Present = 1;
                    pUserPageTable[i].m_Entrys[j].m_ReadWriter = 0;
                    pUserPageTable[i].m_Entrys[j].m_PageBaseAddress =
                }
                else if (j > text_len && j <= text_len + data_len) {
                    pUserPageTable[i].m_Entrys[j].m_Present = 1;
```

Console Memory

CDT Build Console [oos]

```
cd ..\targets\img && partcopy kernel.bin 0 13000 c.img 200
copy ..\targets\objs\boot.bin ..\tools\MakeImage\bin\Debug\boot.bin
已复制    1 个文件。
copy ..\targets\objs\kernel.bin ..\tools\MakeImage\bin\Debug\kernel.bi
已复制    1 个文件。
copy ..\targets\img\c.img ..\tools\MakeImage\bin\Debug\c.img
已复制    1 个文件。
cd ..\tools\MakeImage\bin\Debug && build.exe c.img boot.bin kernel.bin
copy ..\tools\MakeImage\bin\Debug\c.img "..\targets\UNIXV6++"\c.img
已复制    1 个文件。

**** Build Finished ****
```

2. 能成功运行