同济大学计算机系

操作系统实验报告

实验内容 去除 UNIX V6++的相对虚实地址映射表

学　　号 2251745

姓　　名 张宇

专　　业 计算机科学与技术

授课老师 方钰

一、完成实验 4.2

1.1 修改后的构建页表代码如下：

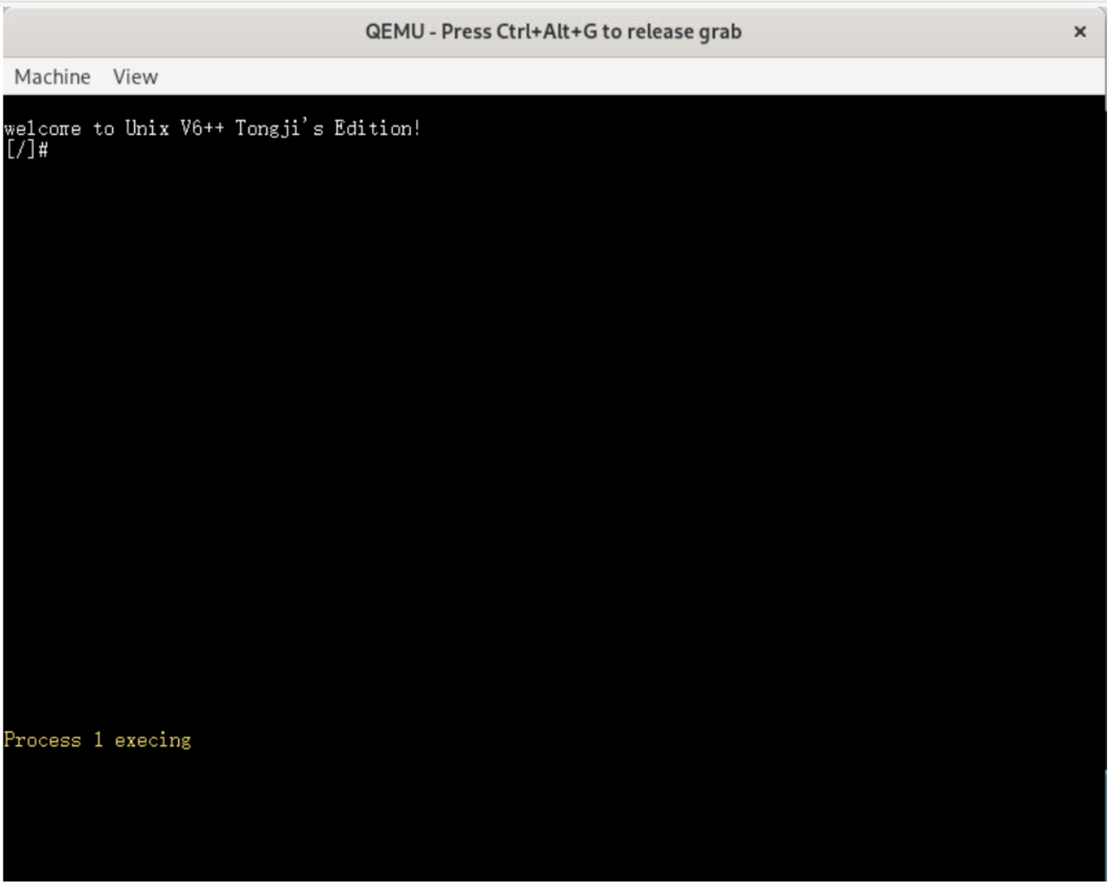
①计算各段起始偏移量和长度

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

```
void MemoryDescriptor::NMapToPageTable()
{
    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;
    }
    // tstart_index对应代码段没有起始偏移量， dstart_index对应数据段和p_addr偏移1个页框号
    unsigned int tstart_index = 0, dstart_index = 1;

    // 计算各个段对应的页框数
    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;

    // 给数据段页框计数
    unsigned int data_index = 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;   //先清0

            // 只刷第2个用户页表
            if (i == 1)
            {
                /* 只读属性表示正文段页，以pText->x_caddr为内存起始地址 */
                if (j >= 1 && j <= text_len) // j==0 为runtime预留
                {
                    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);
                }
                /* 读写属性表示数据段对应的页，以p_addr为内存起始地址 */
                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 = (data_index++) + dstart_index + (u.u_procp->p_addr >> 12);
                }
                /* 堆栈段，从末尾 - stack_len开始  */
                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 = (data_index++) + dstart_index + (u.u_procp->p_addr >> 12);
                }
            }
        }
    }
    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();
}
```

1.2 重新编译并运行结果



二、完成实验 4.3

2.1 Initialize()

修改后的代码如下，因为在这个函数中，将返回值赋为 NULL，不再申请装相对虚实映射表的空间，所以在其他函数中调用初始化的代码时不会产生错误，故不需要修改

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

    // /* m_UserPageTableArray 需要把 AllocMemory()返回的物理内存地址 +
0xC0000000 */
    // this->m_UserPageTableArray =
(PageTable*)(kernelPageManager.AllocMemory(sizeof(PageTable) *
USER_SPACE_PAGE_TABLE_CNT) + Machine::KERNEL_SPACE_START_ADDRESS);

    this->m_UserPageTableArray = NULL;
}
```

2.2 Release()

修改后的代码如下，因为在初始化的函数中赋值为 NULL，所以这个释放空间的函数不需要进行修改

```
void MemoryDescriptor::Release()
{
    KernelPageManager& kernelPageManager =
Kernel::Instance().GetKernelPageManager();
    if ( this->m_UserPageTableArray )
    {
        kernelPageManager.FreeMemory(sizeof(PageTable) *
USER_SPACE_PAGE_TABLE_CNT, (unsigned long)this->m_UserPageTableArray -
Machine::KERNEL_SPACE_START_ADDRESS);
        this->m_UserPageTableArray = NULL;
    }
}
```

2.3 MapEntry ()

修改后的代码如下，因为不需要进行写页表的操作，故将本来映射的代码注释，返回一个无效的页框号即可。

```
unsigned int MemoryDescriptor::MapEntry(unsigned long virtualAddress,
unsigned int size, unsigned long phyPageIdx, bool isReadWrite)
{
    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;
    // unsigned long cnt = ( size + (PageManager::PAGE_SIZE - 1) )/
PageManager::PAGE_SIZE;

    // 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;
}
```

2.4 GetUserPageTableArray ()

修改后的代码如下，因为在初始化的函数中赋值为 NULL，所以这个函数不需要进行修改

```
PageTable* MemoryDescriptor::GetUserPageTableArray()
{
    return this->m_UserPageTableArray;
}
```

2.5 ClearUserPageTable ()

修改后的代码如下，因为在初始化的函数中赋值为 NULL，所以这个清空相对虚

实地址映射表的函数不需要了，全部注释即可。

```
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;
    //  }
    // }

}
```

2.6 EstablishUserPageTable ()

修改后的代码如下，因为在初始化的函数中赋值为 NULL，所以不需要再建立相对虚实地址映射表，存下这些信息直接填页表即可

```
bool MemoryDescriptor::EstablishUserPageTable( unsigned long
textVirtualAddress, unsigned long textSize, unsigned long
dataVirtualAddress, unsigned long dataSize, unsigned long stackSize )
{
    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;
    }
    m_TextSize = textSize;
    m_DataSize = dataSize;
    m_StackSize = stackSize;

    // 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->NMapToPageTable();
    return true;
}
```

三、完成实验 4.4

3.1 对比 User 结构的内容

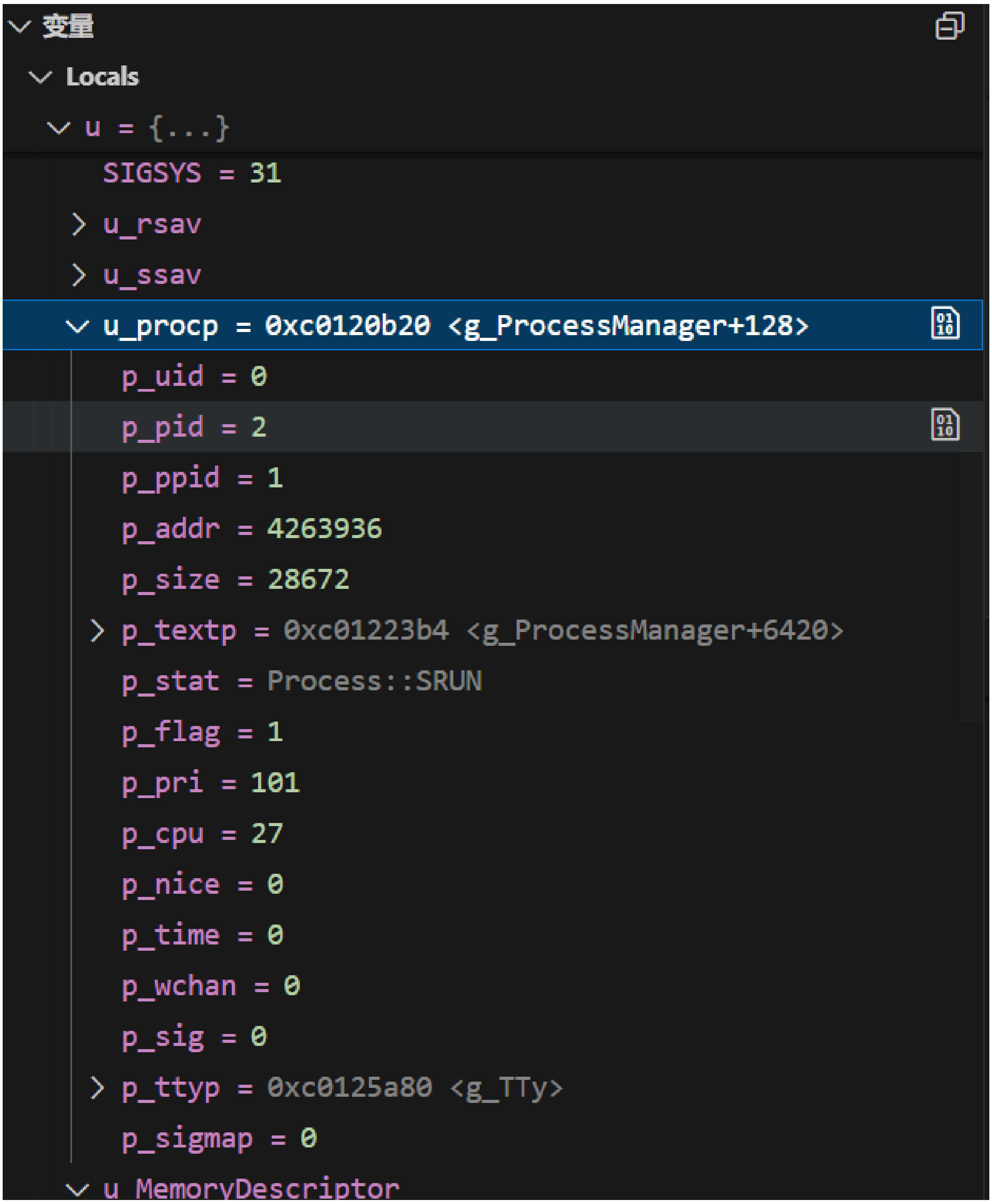
```
-exec x /100xw 0xC03FF000
0xc03ff000:     0xc03fff6c      0xc03fff84      0x00000000      0x00000000
0xc03ff010:     0xc0120b20      0x00000000      0x00401000      0x00003000
0xc03ff020:     0x00404000      0x00005000      0x00001000      0xc03fffcc
0xc03ff030:     0x00000000      0x007ffb98      0x0000000b      0x00000000
0xc03ff040:     0xc03fffec      0x00000000      0x00000000      0x00000000
0xc03ff050:     0x00000000      0x00000000      0x00000000      0x00000000
0xc03ff060:     0x00000000      0x00000000      0x00000000      0x00000000
0xc03ff070:     0x00000000      0x00000000      0x00000000      0x00000000
0xc03ff080:     0x00000000      0x00000000      0x00000000      0x00000000
0xc03ff090:     0x00000000      0x00000000      0x00000000      0x00000000
0xc03ff0a0:     0x00000000      0x00000000      0x00000000      0x00000000
0xc03ff0b0:     0x00000000      0x00000000      0x00000000      0x00000000
```

```
∨ 变量
 ∨ Locals
  ∨ u = {...}
      SIGSYS = 31
    > u_rsav
    > u_ssav
    > u_procp = 0xc0120b20 <g_ProcessManager+128>
    ∨ u_MemoryDescriptor
        USER_SPACE_SIZE = 8388608
        USER_SPACE_PAGE_TABLE_CNT = 2
        USER_SPACE_START_ADDRESS = 0
      > m_UserPageTableArray = 0x0
        m_TextStartAddress = 4198400
        m_TextSize = 12288
        m_DataStartAddress = 4210688
        m_DataSize = 20480
        m_StackSize = 4096
    > u_ar0 = 0xc03fffcc
    > u_arg
    > u_dirp = 0x0
      u_utime = 0
      u_stime = 0
      u_cutime = 0
      u_cstime = 0
> 监视
```

可以看到相对映射表已经被删除了，代码段的起始地址也随之发生改变，除此之外其余的 User 结构均保持不变

3.2 对比 Proc 结构的内容

```
-exec x /100xw 0xc0120b20
0xc0120b20 <g_ProcessManager+128>:    0x00000000    0x00000002    0x00000001    0x00411000
0xc0120b30 <g_ProcessManager+144>:    0x00007000    0xc01223b4    0x00000003    0x00000001
0xc0120b40 <g_ProcessManager+160>:    0x00000065    0x0000001b    0x00000000    0x00000000
0xc0120b50 <g_ProcessManager+176>:    0x00000000    0x00000000    0xc0125a80    0x00000000
0xc0120b60 <g_ProcessManager+192>:    0x00000000    0x00000000    0xffffffff    0x00000000
0xc0120b70 <g_ProcessManager+208>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0120b80 <g_ProcessManager+224>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0120b90 <g_ProcessManager+240>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0120ba0 <g_ProcessManager+256>:    0x00000000    0x00000000    0xffffffff    0x00000000
0xc0120bb0 <g_ProcessManager+272>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0120bc0 <g_ProcessManager+288>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0120bd0 <g_ProcessManager+304>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0120be0 <g_ProcessManager+320>:    0x00000000    0x00000000    0xffffffff    0x00000000
0xc0120bf0 <g_ProcessManager+336>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0120c00 <g_ProcessManager+352>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0120c10 <g_ProcessManager+368>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0120c20 <g_ProcessManager+384>:    0x00000000    0x00000000    0xffffffff    0x00000000
0xc0120c30 <g_ProcessManager+400>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0120c40 <g_ProcessManager+416>:    0x00000000    0x00000000    0x00000000    0x00000000
```



由上图对比可得，只有 p_addr 的值发生了改变，即 ppda 区的物理地址改变，其余值不变

3.3 对比进程代码段的 Text 结构的内容

```
-exec x /100xw 0xc01223b4
0xc01223b4 <g_ProcessManager+6420>:    0x00004738    0x0040e000    0x00003000    0xc011ad9c
0xc01223c4 <g_ProcessManager+6436>:    0x00010001    0x00000000    0x00000000    0x00000000
0xc01223d4 <g_ProcessManager+6452>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc01223e4 <g_ProcessManager+6468>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc01223f4 <g_ProcessManager+6484>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0122404 <g_ProcessManager+6500>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0122414 <g_ProcessManager+6516>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0122424 <g_ProcessManager+6532>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0122434 <g_ProcessManager+6548>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0122444 <g_ProcessManager+6564>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0122454 <g_ProcessManager+6580>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0122464 <g_ProcessManager+6596>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0122474 <g_ProcessManager+6612>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0122484 <g_ProcessManager+6628>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc0122494 <g_ProcessManager+6644>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc01224a4 <g_ProcessManager+6660>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc01224b4 <g_ProcessManager+6676>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc01224c4 <g_ProcessManager+6692>:    0x00000000    0x00000000    0x00000000    0x00000000
0xc01224d4 <g_ProcessManager+6708>:    0x00000000    0x00000000    0x00000000    0x00000000
```

```
p_textp = 0xc01223b4 <g_ProcessManager+6420>
    x_daddr = 18232
    x_caddr = 4251648
    x_size = 12288
    x_iptr = 0xc011ad9c <g_InodeTable+380>
    x_count = 1
    x_ccount = 1
p_stat = Process::SRUN
p_flag = 1
```

由上图对比可得，所有的值均不变