# P05：去除相对虚实地址映射表相关实验

2152118 史君宝

## 任务一、阅读附件中的文档　相对虚实地址映射表出现的所有位置.pptx。读通注释其中出现的所有子程序。

这里并不需要回答，我们跳过这个任务

## 任务二、去除相对虚实地址映射表。

（1）首先我们打开 MemoryDescriptor.cpp 这个文件可以看到下面的内容：

```
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(Pa
    this->m_UserPageTableArray = NULL;
}
```

我们将 this->m_UserPageTableArray 设置为 NULL。

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

```
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;
}
```

我们将这个函数中的内容给注释掉。

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

之后我们将 GetUserPageTableArray 函数的返回值设置为 NULL。

```
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;
        //}
    //}

}
```

之后我们将 ClearUserPageTable 函数中的内容全部注释掉。

```
bool MemoryDescriptor::EstablishUserPageTable( unsigned long textVirtualAddress, unsigned long textSize, unsigne
{
    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->MapToPageTable();
    return true;
```

在这个函数中，我们加入：

```
    m_TextSize = textSize;
    m_DataSize = dataSize;
    m_StackSize = stackSize;
```

并把之后的内容全部注释掉就可以了。

```
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;
    }

    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;   //先清0

            if ( 1 == i )
            {
                /* 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 ( 1 <= i && 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);
                }
                /* 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 = 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();
}
```

这个函数的改动有点大，这里我们就不再赘述了。

（2）去除相对虚实地址映射表的必要性

首先我们需要理解为什么我们能够轻松的实现相对虚实地址映射表，这是因为这本就是一个多余的过程，我们能够轻松地不借助相对虚实　映射表，将需要的变量全部找到。相当于少了一个中转站，因此它

有下面的必要性：

1. 会减少地址转换的开销，使得每个进程在运行过程中占用内存更少 。

2. 满足实时性要求，在系统中，能够确保任务及时响应。去掉虚实地址映射表可以加快进程切换速度，在某些情况下，简化的内存管理方案可能更有利于实时性的保证。

# 任务三、去除相对虚实地址映射表的指针

（1）在 ProcessManager.cpp 程序中我们还需要进行修改：

```
/* 将父进程的用户态页表指针m_UserPageTableArray备份至pgTable */
//PageTable* pgTable = u.u_MemoryDescriptor.m_UserPageTableArray;
u.u_MemoryDescriptor.Initialize();
/* 父进程的相对地址映照表拷贝给子进程，共两张页表的大小 */
//if ( NULL != pgTable )
//{
    //Utility::MemCopy((unsigned long)pgTable, (unsigned long)u.u_MemoryDescriptor.m_UserPage
//}

//将先运行进程的u区的u_procp指向new process
//这样可以在被复制的时候可以直接复制u_procp的
//地址，在内存不够时，是无法将u区映射到用户区，
//修改u_procp的地址的
u.u_procp = child;
```

```
    u.u_procp = current;
    /*
     * 拷贝进程图像期间，父进程的m_UserPageTableArray指向子进程的相对地址映照表；
     * 复制完成后才能恢复为先前备份的pgTable。
     */
    //u.u_MemoryDescriptor.m_UserPageTableArray = pgTable;
    //Diagnose::Write("End NewProc()\n");
    return 0;
}
```

（2）我们可以开始执行：

在 cmd 中 clean ：

```
D:\UNIX_V6++\oos\tools>clean
del     ..\targets\objs\*.o
del     ..\targets\objs\*.exe
del     ..\targets\objs\*.bin
del     ..\targets\objs\*.sym
del     ..\targets\objs\*.asm
del     ..\targets\img\*.bin
del     ..\targets\img\*.sym
del     ..\targets\img\*.asm
del     "..\targets\UNIXV6++"\c.img

D:\UNIX_V6++\oos\tools>
```

Build all：

```
D:\UNIX_V6++\oos\tools>all
make --directory=boot
make[1]: Entering directory `D:/UNIX_V6++/oos/src/boot'
nasm -f bin boot.s -o ..\..\targets\objs\boot.bin
nasm -f elf sector2.s -o ..\..\targets\objs\sector2.bin
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -c support
.c -o ..\..\targets\objs\support.o
make[1]: Leaving directory `D:/UNIX_V6++/oos/src/boot'
make --directory=kernel
make[1]: Entering directory `D:/UNIX_V6++/oos/src/kernel'
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
ude" -c main.cpp -o ..\..\targets\objs\main.o
main.cpp: In function `int main0()':
main.cpp:92: warning: control reaches end of non-void function
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
ude" -c Kernel.cpp -o ..\..\targets\objs\kernel.o
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
ude" -c Video.cpp -o ..\..\targets\objs\video.o
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
ude" -c Utility.cpp -o ..\..\targets\objs\utility.o
make[1]: Leaving directory `D:/UNIX_V6++/oos/src/kernel'
make --directory=machine
make[1]: Entering directory `D:/UNIX_V6++/oos/src/machine'
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
ude" -c Chip8253.cpp -o ..\..\targets\objs\chip8253.o
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
ude" -c Chip8259A.cpp -o ..\..\targets\objs\chip8259A.o
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
```

```
ude" -c SystemCall.cpp -o ..\..\targets\objs\systemcall.o
SystemCall.cpp: In static member function `static int SystemCall::Sys_Getppid()':
SystemCall.cpp:720: warning: converting of negative value `-0x000000001' to `unsigned int'
SystemCall.cpp:717: warning: unused variable 'i'
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
ude" -c DiskInterrupt.cpp -o ..\..\targets\objs\diskinterrupt.o
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
ude" -c KeyboardInterrupt.cpp -o ..\..\targets\objs\keyboardinterrupt.o
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
ude" -c TimeInterrupt.cpp -o ..\..\targets\objs\timeinterrupt.o
make[1]: Leaving directory `D:/UNIX_V6++/oos/src/interrupt'
make --directory=mm
make[1]: Entering directory `D:/UNIX_V6++/oos/src/mm'
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
ude" -c Allocator.cpp -o ..\..\targets\objs\allocator.o
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
ude" -c PageManager.cpp -o ..\..\targets\objs\pagemanager.o
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
ude" -c KernelAllocator.cpp -o ..\..\targets\objs\kernelallocator.o
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -fcheck-ne
w -I"..\include" -c New.cpp -o ..\..\targets\objs\new.o
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
ude" -c SwapperManager.cpp -o ..\..\targets\objs\swappermanager.o
make[1]: Leaving directory `D:/UNIX_V6++/oos/src/mm'
make --directory=proc
make[1]: Entering directory `D:/UNIX_V6++/oos/src/proc'
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
ude" -c Process.cpp -o ..\..\targets\objs\process.o
g++ -Wall -O0 -g -nostartfiles -nostdlib -fno-builtin -fno-rtti -fno-exceptions -nostdinc -I"..\incl
```

之后是 run：

```
[/]#ls
Directory '/':
dev     Shell.exe       bin     demos   etc     usr     var
[/]#cd bin
[/bin]#ls
Directory '/bin':
cat     cat.exe cat1.exe        cp      cp.exe  cpfile.exe      date    date.exe
        echo    echo.exe        forks.exe       getppid.exe     ls      ls.exe
malloc.exe      mkdir   mkdir.exe       newsig.exe      perf    perf.exe
rm      rm.exe  showStack.exe   shutdown        shutdown.exe    sig.exe sigTest.
exe     stack.exe       test.exe        trace   trace.exe
[/bin]#getppid.exe
This is Process 4# speaking...
My parent process ID is: 1
[/bin]#
end sleep
Process 4 (Status:5)  end wait
```