操作系统　进程管理、内存管理项目

补、Unix V6++内存管理关键数据结构

同济大学计算机系

一、进程的内存描述符

```
class User
{
    ……
    MemoryDescriptor u_MemoryDescriptor;
    ……
}
```

```
class MemoryDescriptor
{
        PageTable*        m_UserPageTableArray;

        unsigned long     m_TextStartAddress;
        unsigned long     m_TextSize;
        unsigned long     m_DataStartAddress;
        unsigned long     m_DataSize;
        unsigned long     m_StackSize;
}
```

```
MemoryDescriptor& md = u.u_MemoryDescriptor;
```

二、页表

base	u/s	r/w	p

struct PageTableEntry **PTE**

```
struct PageTableEntry
{
unsigned char        m_Present : 1;
unsigned char        m_ReadWriter : 1;
unsigned char        m_UserSupervisor : 1;
unsigned char        m_WriteThrough : 1;
unsigned char        m_CacheDisabled : 1;
unsigned char        m_Accessed : 1;
unsigned char        m_Dirty : 1;
unsigned char        m_PageTableAttribueIndex : 1;
unsigned char        m_GlobalPage : 1;
unsigned char        m_ForSystemUser : 3;
unsigned int         m_PageBaseAddress : 20;
}__attribute__((packed));
```

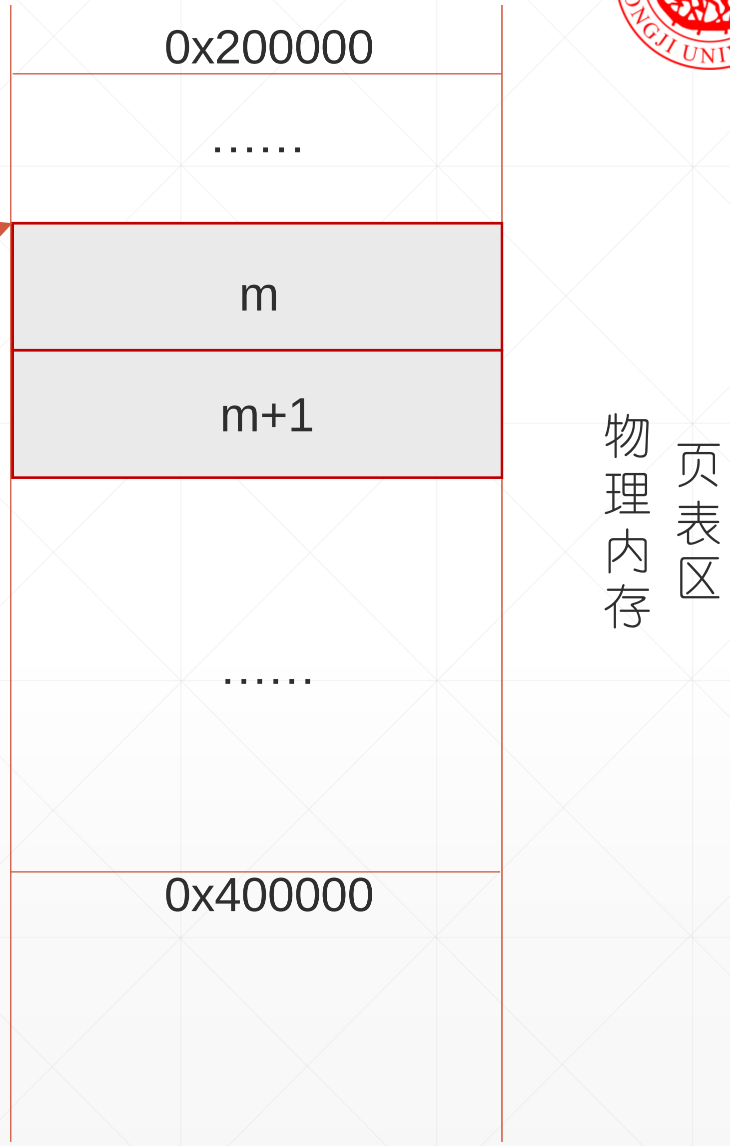
class PageTable 页表

```
class PageTable
{
public:
static const unsigned int ENTRY_CNT_PER_PAGETABLE = 1024;
static const unsigned int SIZE_PER_PAGETABLE_MAP = 0x400000;

public:
PageTable();
~PageTable();

public:
PageTableEntry m_Entrys[ENTRY_CNT_PER_PAGETABLE];
};
```

1024个元素的PTE数组

三、进程的相对虚实地址映射表

```
class MemoryDescriptor
{
        PageTable*        m_UserPageTableArray;

        unsigned long     m_TextStartAddress;
        unsigned long     m_TextSize;
        unsigned long     m_DataStartAddress;
        unsigned long     m_DataSize;
        unsigned long     m_StackSize;
}
```



引用相对表：
u.u_MemoryDescriptor.m_UserPageTableArray

四、系统页表

```
class Machine
{
        ……
        PageDirectory*    m_PageDirectory;      // 0xC0200000
        PageTable*        m_KernelPageTable;   // 0xC0201000
        PageTable*        m_UserPageTable;     // 0xC0202000
}
```

4.1 页目录的初始化

```
void Machine::InitPageDirectory()
{
        PageDirectory* pPageDirectory = (PageDirectory*)( PAGE_DIRECTORY_BASE_ADDRESS +
                                    KERNEL_SPACE_START_ADDRESS);     // 页目录首地址

        /* 填写页目录的第0x300(768)项，使线性地址0xC0000000-0xC0400000映射到物理内存0-4M */
        unsigned int kPageTableIdx = KERNEL_SPACE_START_ADDRESS /
                                    PageTable::SIZE_PER_PAGETABLE_MAP;   // 0xC0000000 / 0x400000
        pPageDirectory->m_Entrys[kPageTableIdx].m_UserSupervisor = 0;      // 核心态
        pPageDirectory->m_Entrys[kPageTableIdx].m_Present = 1;
        pPageDirectory->m_Entrys[kPageTableIdx].m_ReadWriter = 1;
        pPageDirectory->m_Entrys[kPageTableIdx].m_PageTableBaseAddress =
                                    KERNEL_PAGE_TABLE_BASE_ADDRESS >> 12;
```

页目录（0x200）
0x201 S RW P

4.2 系统核心页表的初始化

```
class Machine
{
        ……
        PageDirectory*   m_PageDirectory;      // 0xC0200000
        PageTable*       m_KernelPageTable;    // 0xC0201000
        PageTable*       m_UserPageTable;      // 0xC0202000
}
```

```
// pPageTable，0xC0201000，核心页表首地址
PageTable* pPageTable = (PageTable*)(KERNEL_PAGE_TABLE_BASE_ADDRESS + KERNEL_SPACE_START_ADDRESS);

// 建立核心空间的地址映射关系：[0xC0000000，0xC0400000] 映射至 [0x00000000，0x00400000]
for ( unsigned int i = 0; i < PageTable::ENTRY_CNT_PER_PAGETABLE; i++ )
{
        pPageTable->m_Entrys[i].m_UserSupervisor = 0;
        pPageTable->m_Entrys[i].m_Present = 1;
        pPageTable->m_Entrys[i].m_ReadWriter = 1;
        pPageTable->m_Entrys[i].m_PageBaseAddress = i;
}

this->m_PageDirectory = pPageDirectory;
this->m_KernelPageTable = pPageTable;
}
```

页目录（0x200）
0x201 S RW P

核心页表（0x201）
0 S RW P
1 S RW P
……
……
1023 S RW P

4.3 设置系统用户页表

```
void Machine::InitUserPageTable()
{
        PageDirectory* pPageDirectory = this->m_PageDirectory;    // 页目录首地址
        PageTable* pUserPageTable =                            // 0xC0202000，系统用户页表首地址
                (PageTable*)(USER_PAGE_TABLE_BASE_ADDRESS + KERNEL_SPACE_START_ADDRESS);

        unsigned int idx = USER_PAGE_TABLE_BASE_ADDRESS >> 12;      // 0x202
        for ( unsigned int j = 0; j < USER_PAGE_TABLE_CNT; j++, idx++ )      // 2
        {
                pPageDirectory->m_Entrys[j].m_UserSupervisor = 1;
                pPageDirectory->m_Entrys[j].m_Present = 1;
                pPageDirectory->m_Entrys[j].m_ReadWriter = 1;
                pPageDirectory->m_Entrys[j].m_PageTableBaseAddress = idx;

                for ( unsigned int i = 0; i < PageTable::ENTRY_CNT_PER_PAGETABLE; i++ )
                {
                        pUserPageTable[j].m_Entrys[i].m_UserSupervisor = 1;
                        pUserPageTable[j].m_Entrys[i].m_Present = 1;
                        pUserPageTable[j].m_Entrys[i].m_ReadWriter = 1;
                        pUserPageTable[j].m_Entrys[i].m_PageBaseAddress = 0x00000 + i +j * 1024;
                }
        }
        this->m_UserPageTable = pUserPageTable;
}
```

页目录（0x200）
0x202 U RW P
0x203 U RW P
0x201 S RW P

用户页表（0x202，0x203）
…… U RW P
……
……
……
……
……
……
……
……
……

五、进程切换时，将新选中进程PPDA区所在的物理页框填入核心态页表，1023#PTE

```
class Machine
{
        PageDirectory*    m_PageDirectory;       // 0xC0200000
        PageTable*        m_KernelPageTable;  // 0xC0201000
        PageTable*        m_UserPageTable;     // 0xC0202000
}
```

base	u/s	r/w	p

```
#define SwtchUStruct(p) \
        Machine::Instance().GetKernelPageTable().m_Entrys[Kernel::USER_PAGE_INDEX].m_PageBaseAddress \
                = (p)->p_addr / PageManager::PAGE_SIZE; \
        FlushPageDirectory();
```

MapToPageTable( )，用进程的相对表写系统用户页表

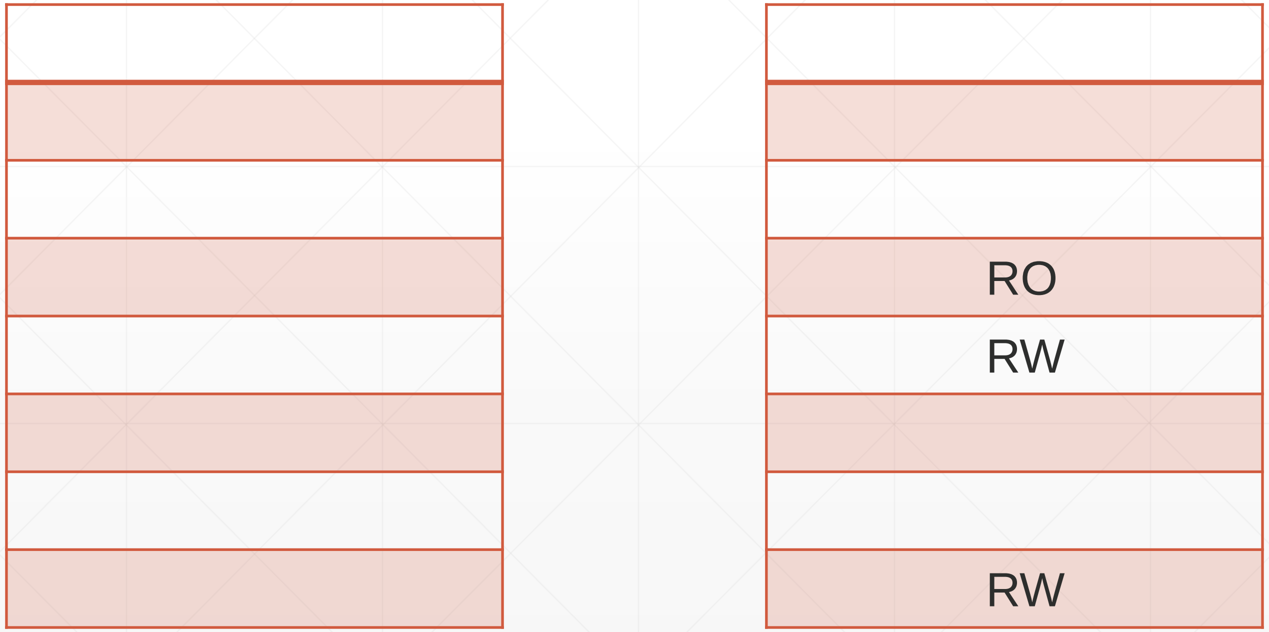
系统用户页表

`pUserPageTable`

`Machine::Instance().GetUserPageTableArray()`

进程的相对表

`u.u_MemoryDescriptor.m_UserPageTableArray`



1、清 0 系统用户页表 （ P=0 ）

2、遍历相对表，对所有 P==1的表项，写系统页表

- RO 表项，base + 代码段起始页框号
- RW 表项，base + 可交换部分起始页框号

3、写 0# PTE：0 U RW P

```
class Machine
{
        PageDirectory*    m_PageDirectory;       // 0xC0200000
        PageTable*        m_KernelPageTable;  // 0xC0201000
        PageTable*        m_UserPageTable;      // 0xC0202000
}
```

```
void MemoryDescriptor::MapToPageTable()
{
        User& u = Kernel::Instance().GetUser();

        if(u.u_MemoryDescriptor.m_UserPageTableArray == NULL)
                return;

        PageTable* pUserPageTable = Machine::Instance().GetUserPageTableArray();
        unsigned int textAddress = 0;
        if ( u.u_procp->p_textp != NULL )
        {
                textAddress = u.u_procp->p_textp->x_caddr;    // 代码段起始页框号
        }
```

```
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;
                        pUserPageTable[i].m_Entrys[j].m_ReadWriter = this->m_UserPageTableArray[i].m_Entrys[j].m_ReadWriter;
                        pUserPageTable[i].m_Entrys[j].m_PageBaseAddress =
                                this->m_UserPageTableArray[i].m_Entrys[j].m_PageBaseAddress + (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 ( 1 == this->m_UserPageTableArray[i].m_Entrys[j].m_ReadWriter )
                {
                        pUserPageTable[i].m_Entrys[j].m_Present = 1;
                        pUserPageTable[i].m_Entrys[j].m_ReadWriter = this->m_UserPageTableArray[i].m_Entrys[j].m_ReadWriter;
                        pUserPageTable[i].m_Entrys[j].m_PageBaseAddress =
                                this->m_UserPageTableArray[i].m_Entrys[j].m_PageBaseAddress + (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();
}
```