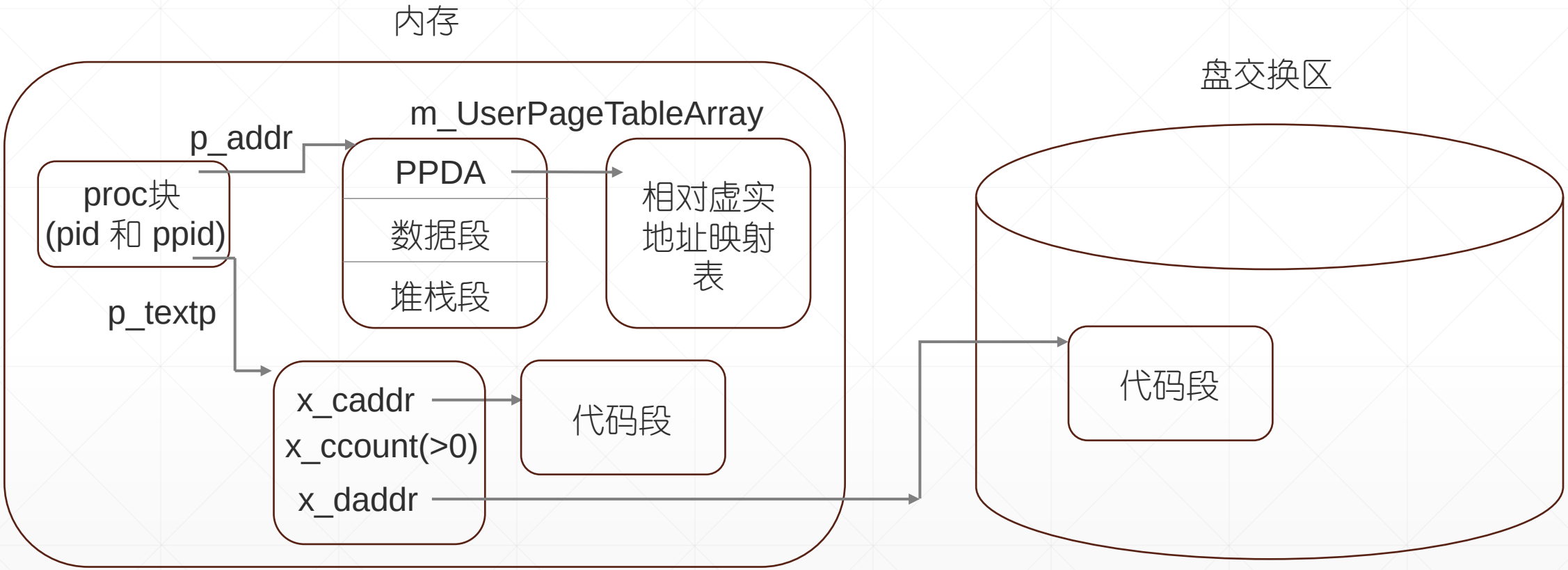
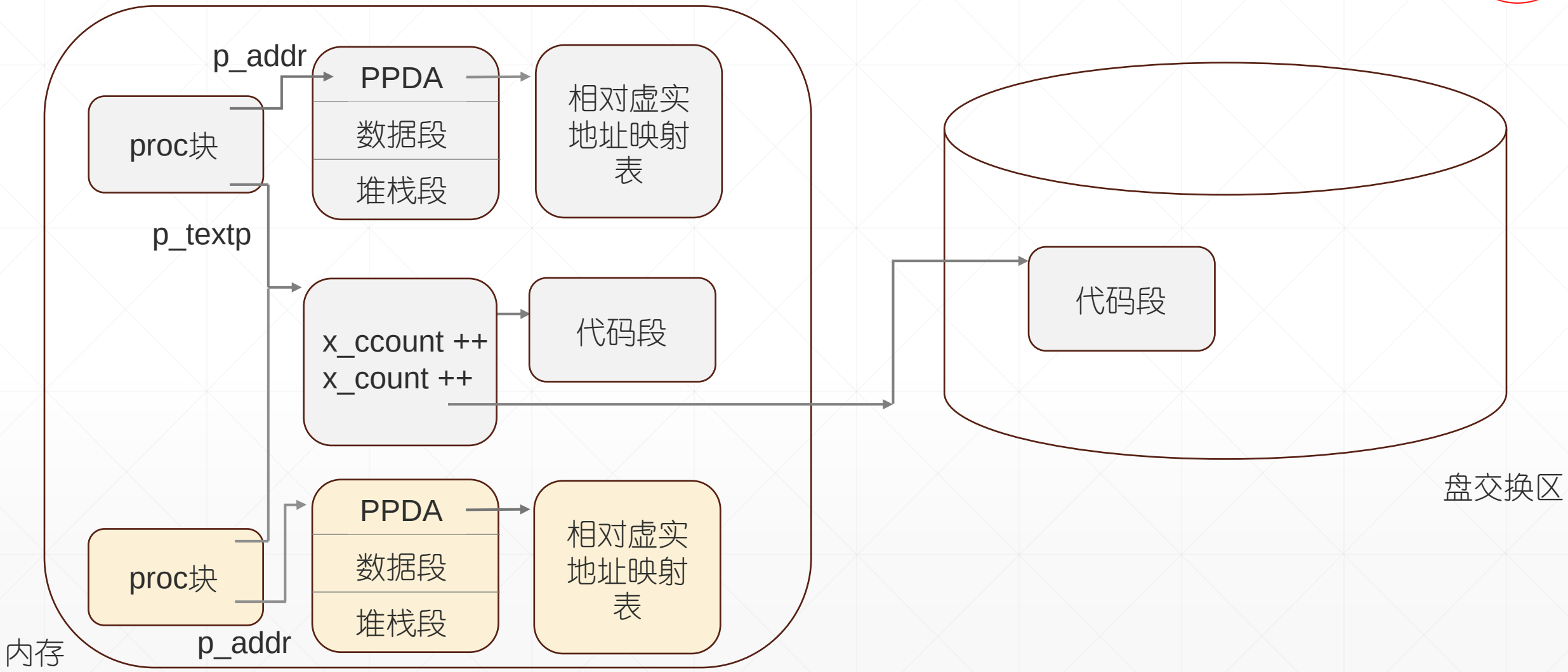
操作系统　第四章　进程管理

4.5 fork系统调用

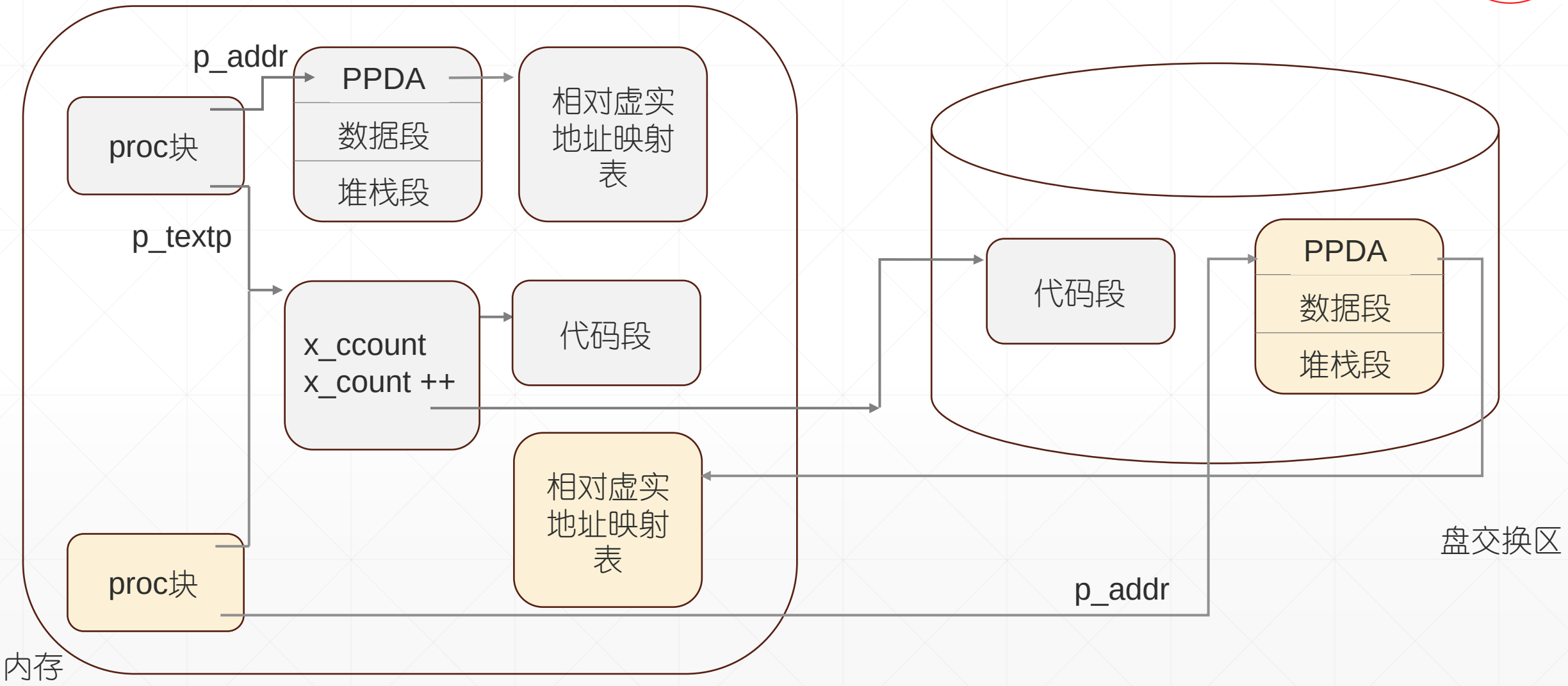
同济大学计算机系


内存
m_UserPageTableArray
p_addr
proc块
(pid 和 ppid)
PPDA
数据段
堆栈段
相对虚实
地址映射
表
p_textp
x_caddr
x_ccount(>0)
x_daddr
代码段
盘交换区
代码段

创建在内存中的子进程图像



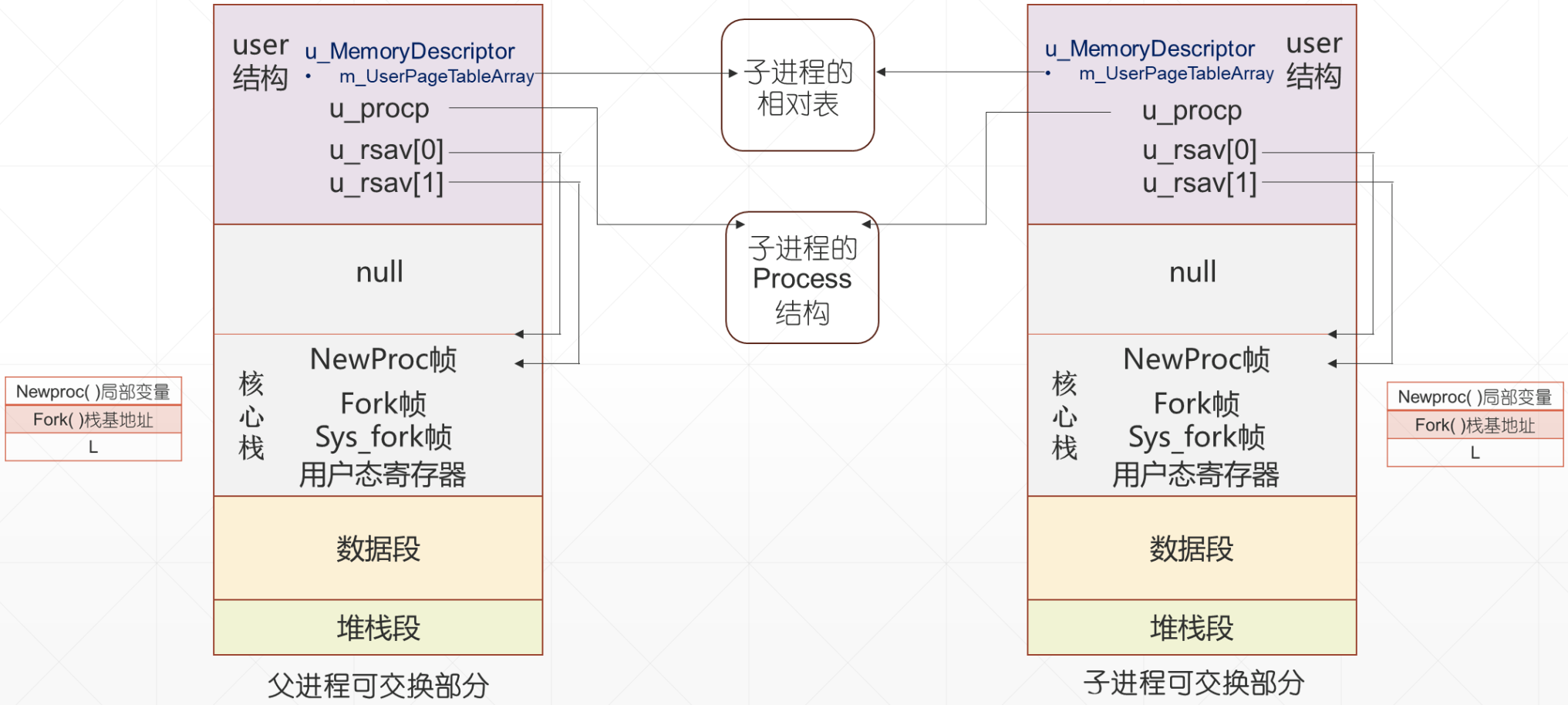
创建在盘交换区上的子进程图像



复制给子进程的Process结构

```
void Process::Clone(Process& proc)   // 指针 this 和 proc 分别指向父进程和子进程的 Process 结构
{
    User& u = Kernel::Instance().GetUser();

    /* 拷贝父进程 Process 结构中的大部分数据 */
    proc.p_size = this->p_size;
    proc.p_stat = Process::SRUN;
    proc.p_flag = Process::SLOAD;
    proc.p_uid = this->p_uid;
    proc.p_ttyp = this->p_ttyp;
    proc.p_nice = this->p_nice;
    proc.p_textp = this->p_textp;

    proc.p_pid = ProcessManager::NextUniquePid();   // 为子进程分配 pid
    proc.p_ppid = this->p_pid;     // 子进程登记父进程的 pid

    proc.p_pri = 0;     // 确保 child 的优先数较小，先于父进程运行（Unix V6++并无此必要
    proc.p_time = 0;    // 驻留时间清 0
```

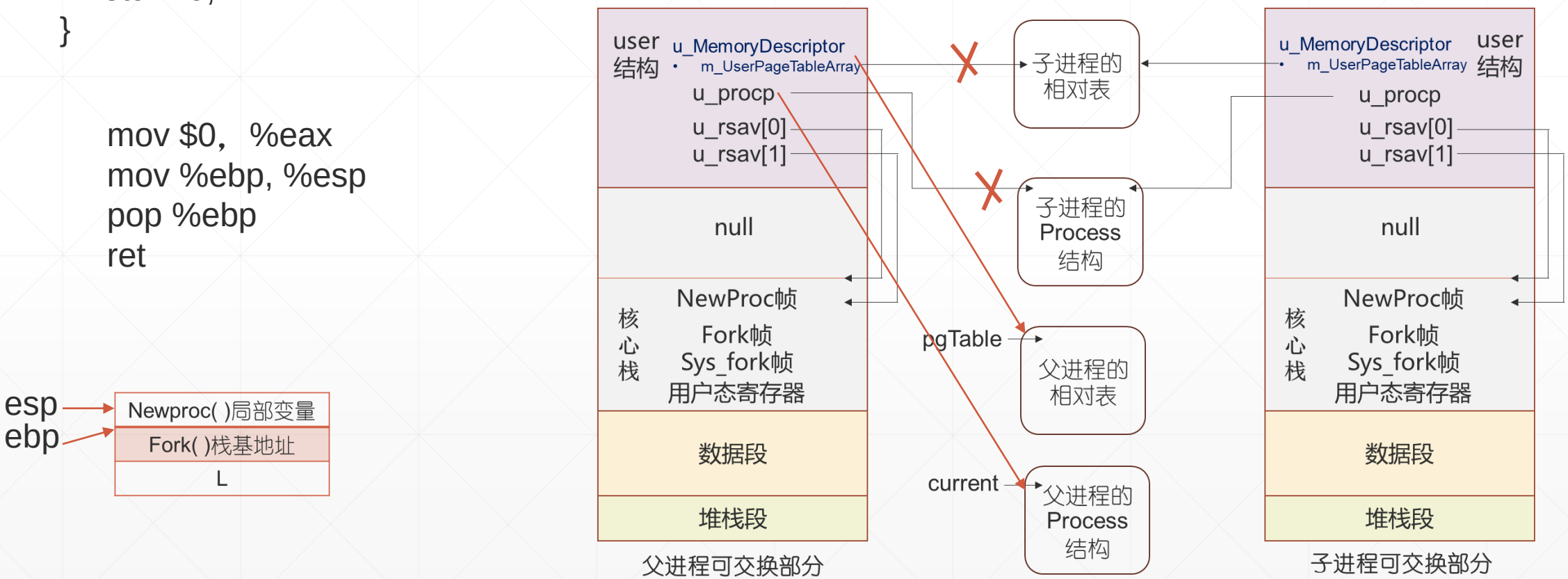
复制给子进程的可交换部分



父进程修改自己的指针后，NewProc返回 0

```
    u.u_procp = current;
    u.u_MemoryDescriptor.m_UserPageTableArray = pgTable;
    return 0;
}
```

```
mov $0，  %eax
mov %ebp, %esp
pop %ebp
ret
```



NewProc( ) → Fork( )

```
if ( this->NewProc() )
{
        u.u_ar0[User::EAX] = 0;
        u.u_cstime = 0;
        u.u_stime = 0;
        u.u_cutime = 0;
        u.u_utime = 0;
}
else
{
        u.u_ar0[User::EAX] = child->p_pid;
}

return;
}
```

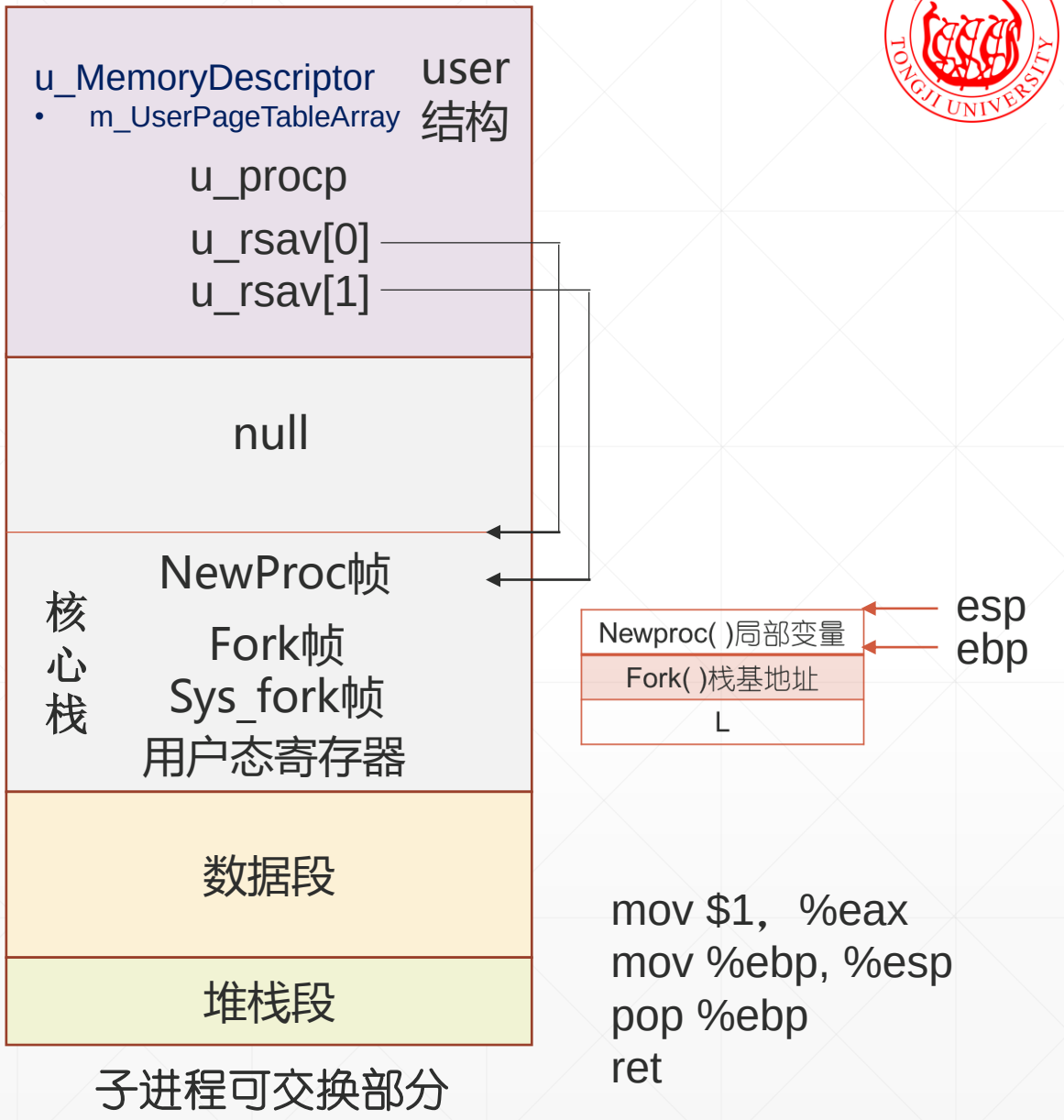
```
    call NewProc
L: cmp %eax，0
    je fatherBranch
childBranch：
        ……
    jmp exit
fatherBranch：
        ……
exit：
```

父进程 fork 系统调用的返回值是 子进程pid

子进程被 Swtch( ) 选中

```
int ProcessManager::Swtch()
{

    User& u = Kernel::Instance().GetUser();
    SaveU(u.u_rsav);

    Process* procZero = &process[0];
    X86Assembly::CLI();
    SwtchUStruct(procZero);
    RetU();
    X86Assembly::STI();

    Process* selected = Select();

    X86Assembly::CLI();
    SwtchUStruct(selected);
L:  RetU();
    X86Assembly::STI();
    User& newu = Kernel::Instance().GetUser();
    newu.u_MemoryDescriptor.MapToPageTable();
    ……
    return 1;
}
```

子进程被 Swtch 选中，RetU 从 u_rsav 中恢复 esp 和 ebp。建立地址映射关系后，退栈 return 1。。。实际上执行的是从NewProc，系统调用返回的工作。。。返回值是 1。



Swtch( ) → Fork( )

```
if ( this->NewProc() )
{
        u.u_ar0[User::EAX] = 0;
        u.u_cstime = 0;
        u.u_stime = 0;
        u.u_cutime = 0;
        u.u_utime = 0;
}
else
{
        u.u_ar0[User::EAX] = child->p_pid;
}

return;
}
```

```
    call NewProc
L: cmp %eax，0
    je fatherBranch
childBranch：
        ……
    jmp exit
fatherBranch：
        ……
exit：
```

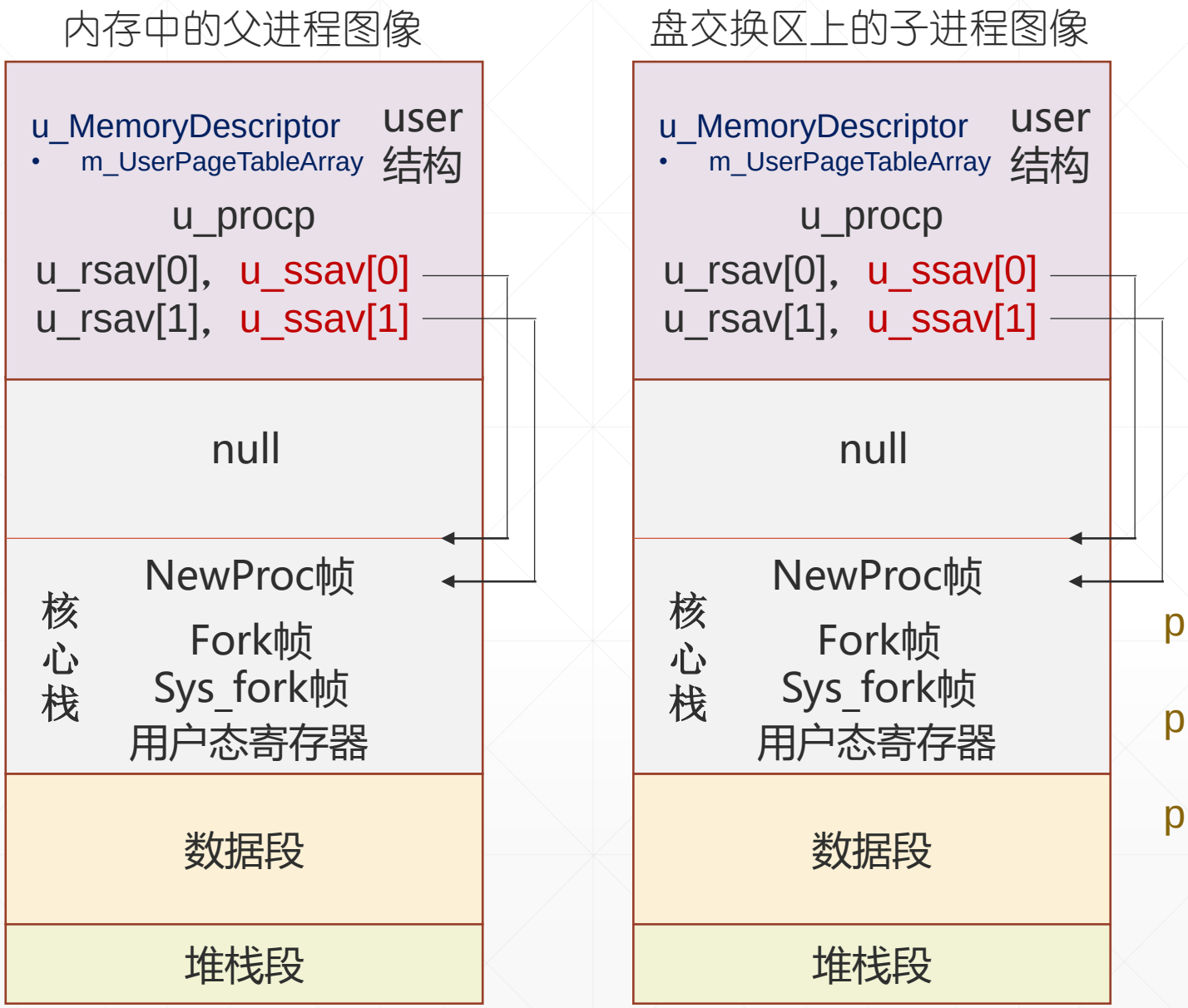
fork 系统调用的返回值是 0

XSwap执行IO操作复制父进程图像

1、NewProc帧的位置复制进u_ssav数组。

2、如果 0# 进程因RunOut入睡，会被唤醒。

3、打 XSwap 标记

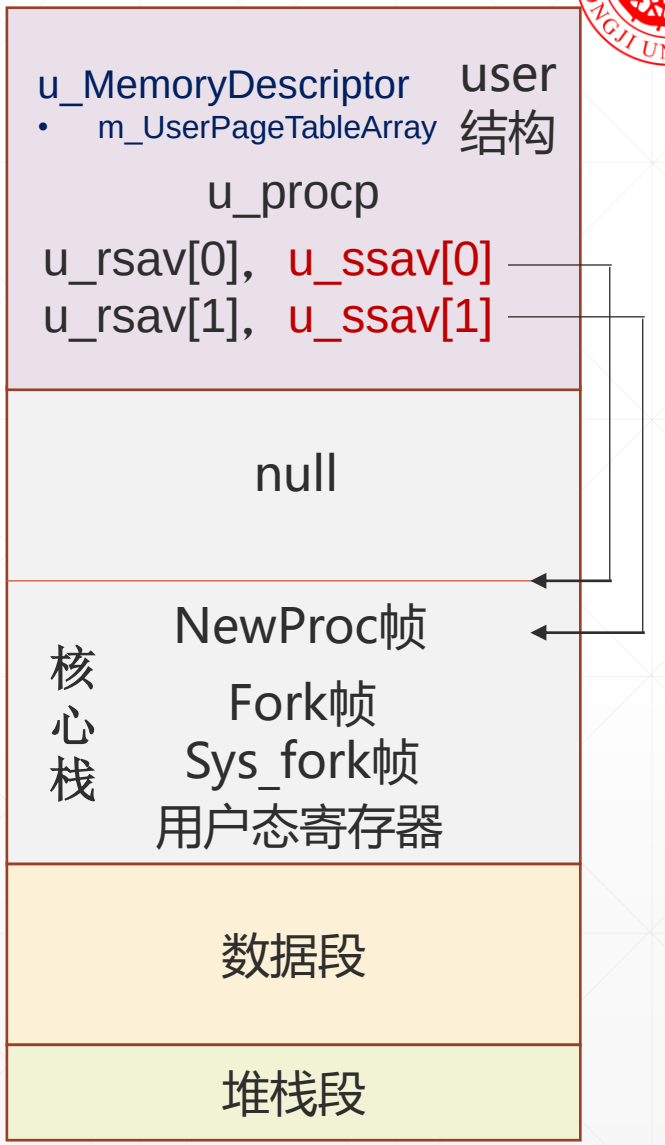


未来，0#进程会将其换入内存

被选中后

```
int ProcessManager::Swtch()
{
        ……
        /* 恢复被保存进程的现场 */
        X86Assembly::CLI();
        SwtchUStruct(selected);
        RetU();
        X86Assembly::STI();

        User& newu = Kernel::Instance().GetUser();
        newu.u_MemoryDescriptor.MapToPageTable();

        if ( newu.u_procp->p_flag & Process::SSWAP )
        {
                newu.u_procp->p_flag &= ~Process::SSWAP;
                aRetU(newu.u_ssav);
        }
        return 1;
}
```

内存中的子进程图像



带 SSWAP 标记，从 u_ssav 中恢复 ESP 和 EBP。退栈，fork系统调用返回用户态