

操作系统 第四章 进程管理

4.4 可执行存储器（内存+盘交换区）的使用



可执行存储器

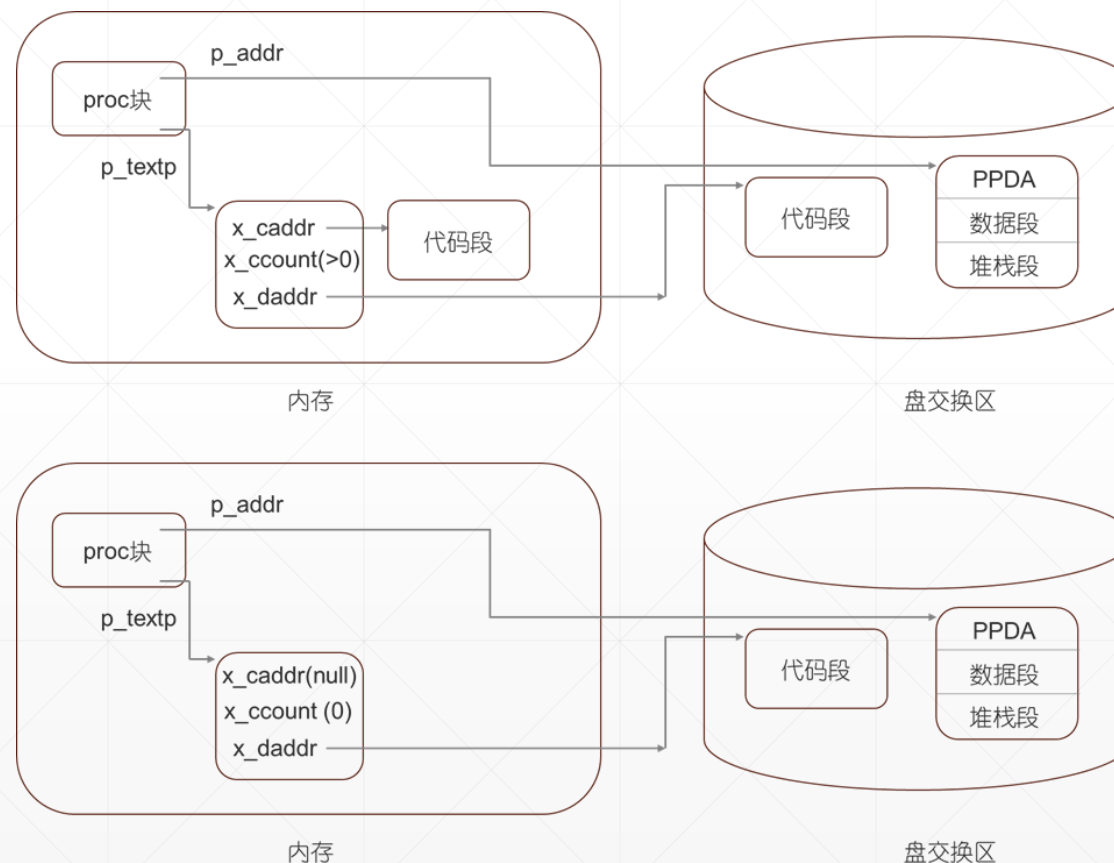
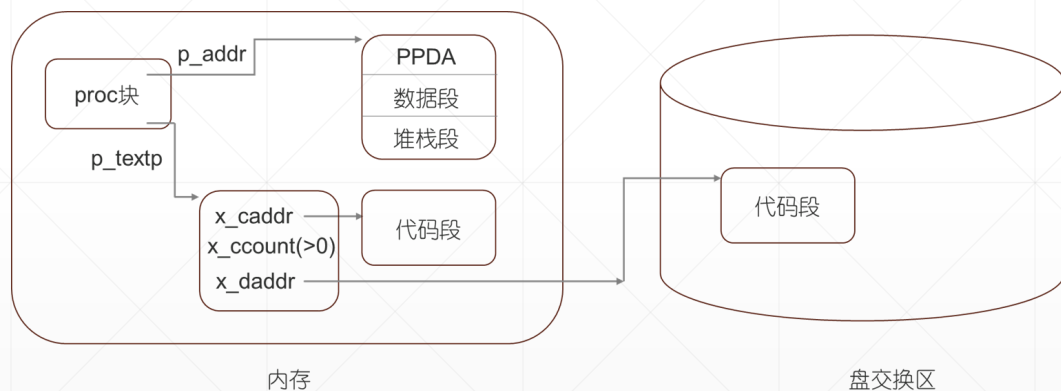
- 可执行存储器是用来存放进程图像的存储单元，包括 内存 和 盘交换区。
- 内存中的进程图像可执行，因为每个单元都有物理地址。
- 盘交换区中的图像不可执行。运行前，必需为其分配内存。



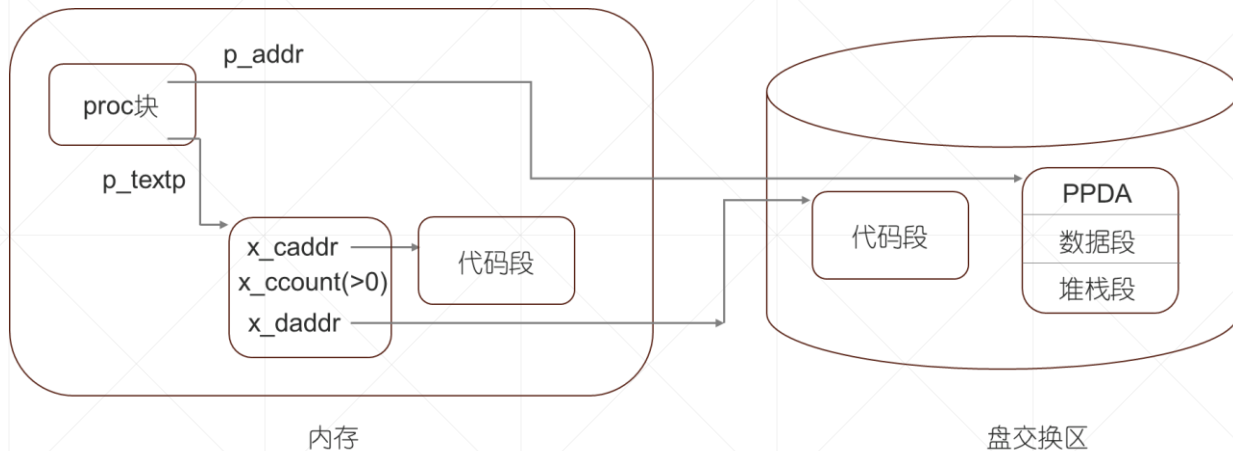
Unix V6++ 进程图像占据的可执行存储器

盘交换区上的进程 **SLOAD = 0**

内存中的进程 **SLOAD = 1**



换入操作



盘交换区上的进程运行前，执行换入操作

图 a

- 1、为可交换部分分配连续内存单元
- 2、启动磁盘IO，复制盘交换区上的 可交换部分
- 3、释放可交换部分占据的盘交换区空间

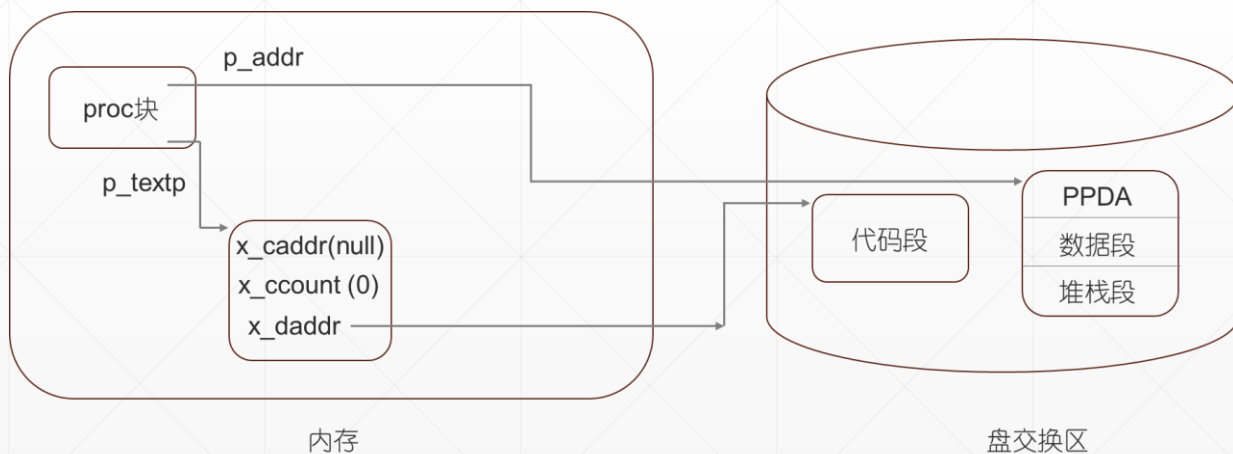
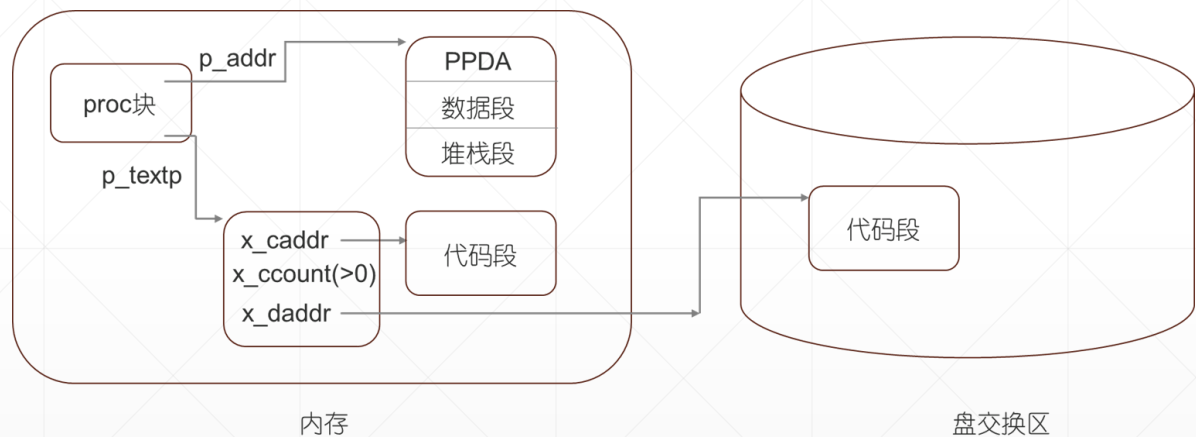


图 b

- 1、为代码段分配连续内存单元，复制代码段图像
- 2、为可交换部分分配连续内存单元，复制可交换部分，释放可交换部分占据的盘交换区空间

Unix V6++系统仅换入就绪进程

换出操作



- 1、为可交换部分分配盘交换区空间
- 2、启动IO，复制可交换部分。释放内存
- 3、`x_ccount - -`，为0，释放代码段占据的内存单元

内存是主存。盘交换区是辅存，能不用尽量不用。
必须使用时，优先换出SWAIT，其次是SSLEEP和SRUN。

可执行存储器管理

```
class ProcessManager
```

```
{
```

```
...
```

```
int RunIn;      // 初值为0。 1表示无法换入盘交换区上的就绪进程
```

```
int RunOut;     // 初值为1。 1表示盘交换区不存在就绪进程
```

```
...
```

```
}
```

Newproc	Expand
Sched	XSwap	Exec
Swap		

Swap, 进程图像 IO。

XSwap, 换出。0#进程 或 现运行进程 实施。

Sched, 换入。由0#进程实施。

Exec, 加载可执行程序时, 盘交换区留一份代码。

Newproc, 创建子进程时, 若内存不足, 调用XSwap将子进程图像创建在盘交换区。

Expand, 进程图像扩展, 若内存不足, 换出当前进程图像。

Sleep(SWAIT), WakeUpAll, Exit。

每个整数秒, 考虑利用已终止进程占用的内存空间。



一、Swap, 启动IO, 在盘交换区 和 内存之间复制进程图像

`bool BufferManager::Swap(int blkno, unsigned long addr, int count, enum Buf::BufFlag flag)`

入口参数:

`blkno`: 盘交换区上的起始扇区号;

`addr`: 内存起始地址;

`count`: 进程图像的长度, 单位是字节;

`flag`: `Buf::B_READ(0x02)`, 磁盘上的进程图像复制进内存;
`Buf::B_WRITE(0x01)`, 反过来。

返回值:

`true`, IO成功; `false`, IO失败;

1、构造IO请求, 启动 IO 操作。

// 随后, 外设芯片 (硬件) 工作, 将磁盘上的图像复制进内存 (或者反过来)。IO 完成后送出磁盘中断信号

2、入睡, 同步等待IO操作结束。

`u.u_procp->Sleep (....., ProcessManager::PSWP)` // 换入、换出操作, 入睡优先数 -100

3、返回



二、XSwap, 换出内存中的进程

```
void ProcessManager::XSwap( Process* pProcess, bool bFreeMemory, int size )
```

- 1、将进程pProcess内存中的可交换区复制到盘交换区。
 - bFreeMemory为true, 复制完成后释放内存中的可交换部分; false选项用于子进程创建, 复制完成后内存中的可交换部分不释放, 供父进程使用, 子进程用盘交换区复本。
 - size是进程图像的大小, 0是默认值, 取可交换部分的长度。
- 2、true分支, 若内存无进程引用pProcess的代码段, 释放空间。



```
void ProcessManager::XSwap( Process* pProcess, bool bFreeMemory, int size )
{
    if ( 0 == size)
    {
        size = pProcess->p_size;
    }

    /* 为进程图像分配盘交换区空间。blkno是起始扇区号 */
    int blkno = Kernel::Instance().GetSwapperManager().AllocSwap(pProcess->p_size);
    if ( 0 == blkno )
    {
        Utility::Panic("Out of Swapper Space");
    }

    if ( pProcess->p_textp != NULL )
    {
        pProcess->p_textp->XccDec(); // text结构, x_ccount --。为0, 释放内存复本。
    }
}
```



```
pProcess->p_flag |= Process::SLOCK;    /* 进程图像复制期间，要上锁。防止同一进程图像被重复换出 */

if ( false == Kernel::Instance().GetBufferManager().Swap(blkno, pProcess->p_addr, size, Buf::B_WRITE) )
{   /* IO操作，可交换部分写磁盘（同步写） */
    Utility::Panic("Swap I/O Error");
}

if ( bFreeMemory )
{   /* 释放内存副本 */
    Kernel::Instance().GetUserPageManager().FreeMemory(size, pProcess->p_addr);
}

pProcess->p_addr = blkno;    /* 修正PCB */
pProcess->p_flag &= ~(Process::SLOAD | Process::SLOCK);
pProcess->p_time = 0;
```



```
class ProcessManager
{
    ...
    int RunIn;    // 初值为0。 1表示无法换入盘交换区上的就绪进程
    int RunOut;   // 初值为1。 1表示盘交换区不存在就绪进程
    ...
}

if ( this->RunOut )
{
    this->RunOut = 0;
    Kernel::Instance().GetProcessManager().WakeUpAll((unsigned long)&RunOut);
}
}
```

后续：XSwap 成功，现运行进程继续执行 NewProc 或 Expand，它会入睡等待，直至0#进程将图像搬入内存。



三、0# 进程和换入操作

- 系统初始化时创建0#进程，执行 Sched()程序。

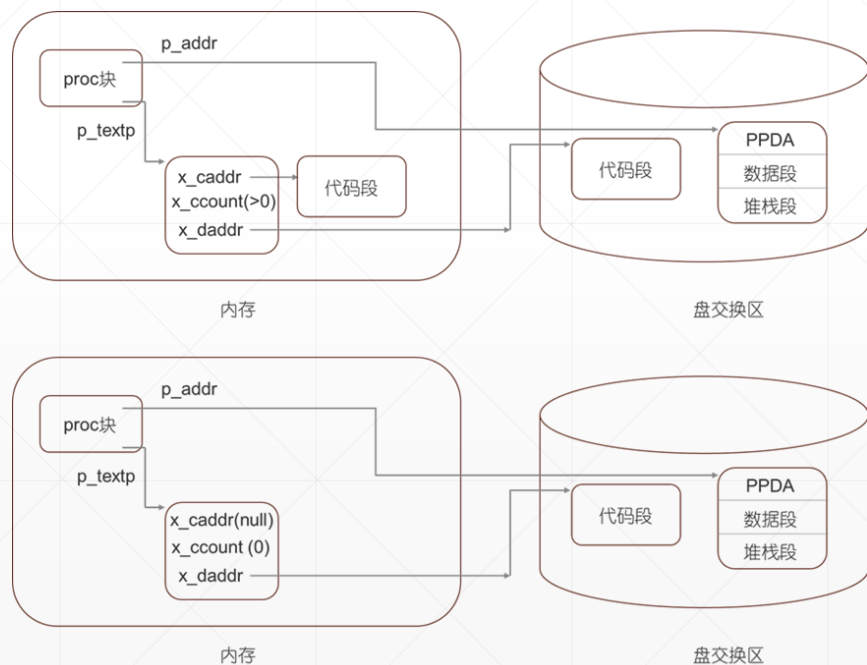
- 初值:

RunIn = RunOut = 0。

```
void ProcessManager::Sched()
{
    User& u = Kernel::Instance().GetUser();
    goto loop;
    .....
loop:
    seconds = -1;
    for ( int i = 0; i < ProcessManager::NPROC; i++ )
    {
        if ( this->process[i].p_stat == Process::SRUN &&
            (this->process[i].p_flag & Process::SLOAD) == 0 &&
            this->process[i].p_time > seconds )
        {
            pSelected = &(this->process[i]);
            seconds = pSelected->p_time; // 盘交换区驻留时长
        }
    }

    /* 如果找着，0#进程睡眠等待第一个 ~SLOAD SRUN */
    if ( -1 == seconds )
    {
        this->RunOut++;
        u.u_procp->Sleep((unsigned long)&RunOut, ProcessManager::PSWP);
        goto loop;
    }
}
```

为盘交换区上的 进程分配内存



```
void ProcessManager::Sched()
```

```
{
```

```
    User& u = Kernel::Instance().GetUser();  
    goto loop;
```

```
sloop:
```

```
    this->RunIn++;
```

```
    u.u_procp->Sleep((unsigned long)&RunIn, ProcessManager::PSWP);
```

```
loop:
```

.....选中盘交换区上的就绪进程 **pSelected**

```
size = pSelected->p_size;
```

```
if ( pSelected->p_textp != NULL && 0 == pSelected->p_textp->x_ccount )  
    size += pSelected->p_textp->x_size;
```

```
desAddress = Kernel::Instance().GetUserPageManager().AllocMemory(size);
```

```
if ( NULL != desAddress )
```

```
{
```

```
    goto found2;
```

```
}
```



found2:

```
BufferManager& bufMgr = Kernel::Instance().GetBufferManager();
if ( pSelected->p_textp != NULL )
{
    Text* pText = pSelected->p_textp;
    if ( pText->x_ccount == 0 )
    {
        /* 复制盘交换区上的正文段 */
        if ( bufMgr.Swap(pText->x_daddr, desAddress, pText->x_size, Buf::B_READ) == false )
            goto err;
        pText->x_caddr = desAddress;
        desAddress += pText->x_size;
    }
    pText->x_ccount++;
}
```

换入操作
(内存有空)

```
/*换入可交换部分*/ if ( bufMgr.Swap(pSelected->p_addr, desAddress, pSelected->p_size, Buf::B_READ) == false )
    goto err;
Kernel::Instance().GetSwapperManager().FreeSwap(pSelected->p_size, pSelected->p_addr);
pSelected->p_addr = desAddress;
pSelected->p_flag |= Process::SLOAD;
pSelected->p_time = 0;
goto loop;
```



```
for ( int i = 0; i < ProcessManager::NPROC; i++ )
{
    if ( this->process[i].p_flag & (Process::SSYS | Process::SLOCK | Process::SLOAD) == Process::SLOAD
        && (this->process[i].p_stat == Process::SWAIT || this->process[i].p_stat == Process::SSTOP) )
        pSelected = &(this->process[i]), goto found1;    // 换出内存中的进程, SWAIT 或 SSTOP
}
```

// 万不得已, 冒着给系统带来抖动的风险换出SSLEEP 或 SRUN。要权衡利弊。

```
if ( seconds < 3 )
    goto sloop;    // PPT13, sleep RunIn
```

内存没空, 要执行对换操作

```
seconds = -1;
for ( int i = 0; i < ProcessManager::NPROC; i++ )
{
    if ( this->process[i].p_flag & (Process::SSYS | Process::SLOCK | Process::SLOAD) == Process::SLOAD
        && (this->process[i].p_stat == Process::SSLEEP || this->process[i].p_stat == Process::SRUN)
        && pSelected->p_time > seconds )
        pSelected = &(this->process[i]), seconds = pSelected->p_time;    // 内存中的就绪进程, 找p_time最大的
}
if ( seconds < 2 )
    goto sloop;    // PPT13, sleep RunIn
```



换出操作 (换出内存中的进程)

found1:

```
X86Assembly::STI();  
pSelected->p_flag &= ~Process::SLOAD;  
this->>XSwap(pSelected, true, 0); // 换出内存中的一个进程  
goto loop; // PPT 12, 再次尝试换入盘交换区上的就绪进程
```




总结: Sched()架构

```
Sched ( )  
{  
    goto loop;  
  
    sloop:  
        RunIn++, Sleep (RunIn); // 等吧, 过会内存会有空的!  
    loop:  
        找 盘交换区 p_time 最大的就绪进程 pSelect;  
        没有。。。好事儿! RunOut++, Sleep( RunOut);  
        否则, 为 pSelect 分配内存  
        内存分配成功 goto found2
```

内存分配不成功, 需要换出内存中的进程, 衡量代价值得, 继续; 不值得, goto sloop。

```
found1:  
    XSwap换出内存中的进程,  
    goto loop 尝试换入目标进程
```

```
found2:  
    换入pSelected;
```

```
goto loop;
```

```
}
```



激活0#进程 (1) 现运行进程低优先权入睡, 放弃CPU前看下 RunIn

```
void Process::Sleep(unsigned long chan, int pri)
{
    if ( pri > 0 )
    {
        this->p_wchan = chan;
        this->p_stat = Process::SWAIT;
        this->p_pri = pri;

        if ( procMgr.RunIn != 0 ) // 盘交换区有 SRUN 等着进内存
        {
            procMgr.RunIn = 0;
            procMgr.WakeUpAll((unsigned long)&procMgr.RunIn);
        }

        Kernel::Instance().GetProcessManager().Swtch(); .....
    }
    .....
}
```

0# 进程换出 SWAIT 的现运行进程, 之后换入盘交换区上的 SRUN 进程

唤醒的0#进程，恢复运行， 执行对换操作

```
void ProcessManager::Sched()
{
    User& u = Kernel::Instance().GetUser();
    goto loop;

sloop:
    this->RunIn++;
    u.u_procp->Sleep((unsigned long)&RunIn, ProcessManager::PSWP);

loop:
    .....选中盘交换区上的就绪进程 pSelected

    为pSelected分配内存，未果

    找到刚刚入睡的SWAIT，换出内存

    goto loop
}
```



激活0#进程 (2) 整数秒, Clock()看下RunIn, 为1时唤醒0#进程

```
void Time::Clock( struct pt_regs* regs, struct pt_context* context )
{
    .....
    if ( procMgr.RunIn != 0 )
    {
        procMgr.RunIn = 0;
        procMgr.WakeUpAll((unsigned long)&procMgr.RunIn);
    }
    .....
}
```

如果前一秒有进程终止，看看腾出来的内存可不可以容纳盘交换区上的SRUN

```
void ProcessManager::Sched()
{
    User& u = Kernel::Instance().GetUser();
    goto loop;

sloop:
    this->RunIn++;
    u.u_procp->Sleep((unsigned long)&RunIn, ProcessManager::PSWP);

loop:
    .....选中盘交换区上的就绪进程 pSelected

    为pSelected分配内存，没准可以装进终止进程原先占据的内存空间

    goto loop
}
```



激活0#进程 (3) 唤醒盘交换区上的就绪进程

```
void ProcessManager::WakeUpAll(unsigned long chan)
{
    for(int i = 0; i < ProcessManager::NPROC; i++)
    {
        if( this->process[i].IsSleepOn(chan) )
            this->process[i].SetRun();
    }
}
```

```
void Process::SetRun()
{
    .....
    this->p_wchan = 0;
    this->p_stat = Process::SRUN;
    if ( this->p_pri < procMgr.CurPri )
        procMgr.RunRun++;
    if ( 0 != procMgr.RunOut && (this->p_flag & Process::SLOAD) == 0 )
    {    // 如果被唤醒的是盘交换区上的进程，把0#进程也叫起来
        procMgr.RunOut = 0;
        procMgr.WakeUpAll((unsigned long)&procMgr.RunOut);
    }
}
```



唤醒的0#进程，
恢复运行，换入
盘交换区上刚刚
被唤醒的进程。

```
void ProcessManager::Sched()
{
    User& u = Kernel::Instance().GetUser();
    goto loop;
    .....
loop:
    seconds = -1;
    for ( int i = 0; i < ProcessManager::NPROC; i++ )
    {
        if ( this->process[i].p_stat == Process::SRUN &&
            (this->process[i].p_flag & Process::SLOAD) == 0 &&
            this->process[i].p_time > seconds )
        {
            pSelected = &(this->process[i]);
            seconds = pSelected->p_time;
        }
    }

    /* 如果没有符合条件的进程，0#进程睡眠等待有需要换入的进程 */
    if ( -1 == seconds )
    {
        this->RunOut++;
        u.u_procp->Sleep((unsigned long)&RunOut, ProcessManager::PSWP);
        goto loop;
    }
}
```