操作系统 2025~2026 期末考纲（上）

计算机拔尖班 2023，国豪 2023　　2025/12/30

一、 进程的表示（Unix V6++系统）

1、 可执行文件格式

2、 进程图像

- 物理空间中的进程图像

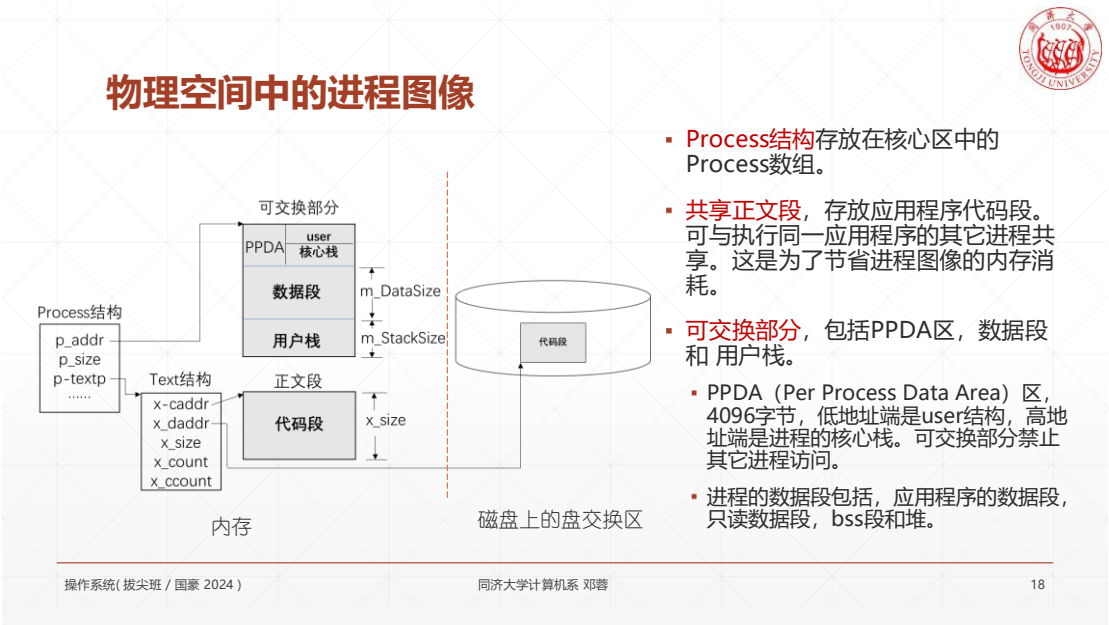


图 1、Unix V6++进程图像（物理空间）

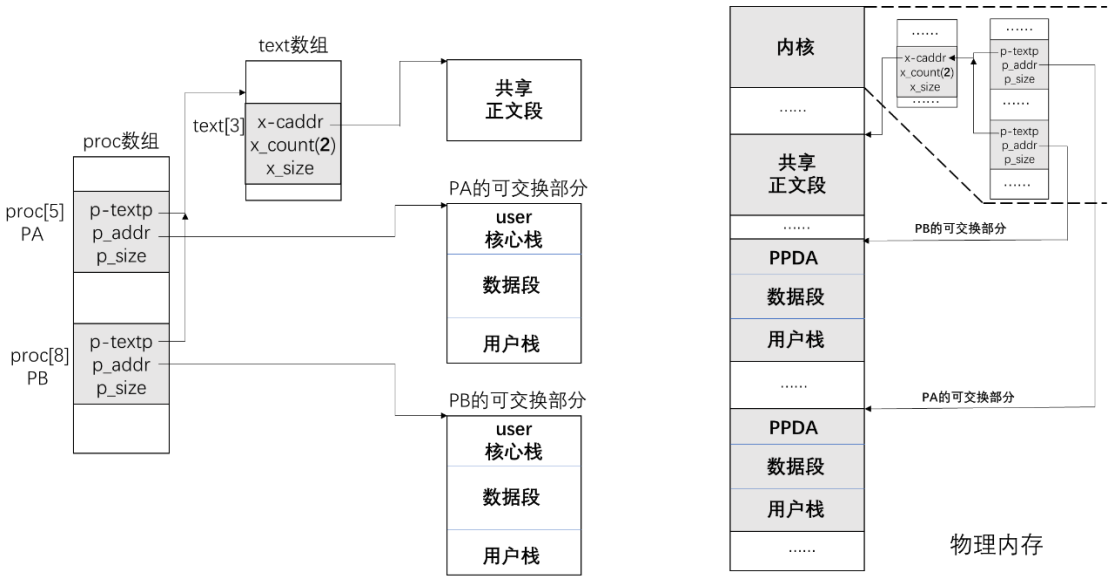


图 2、进程 PA、PB 共享正文段

- 虚空间中的进程图像

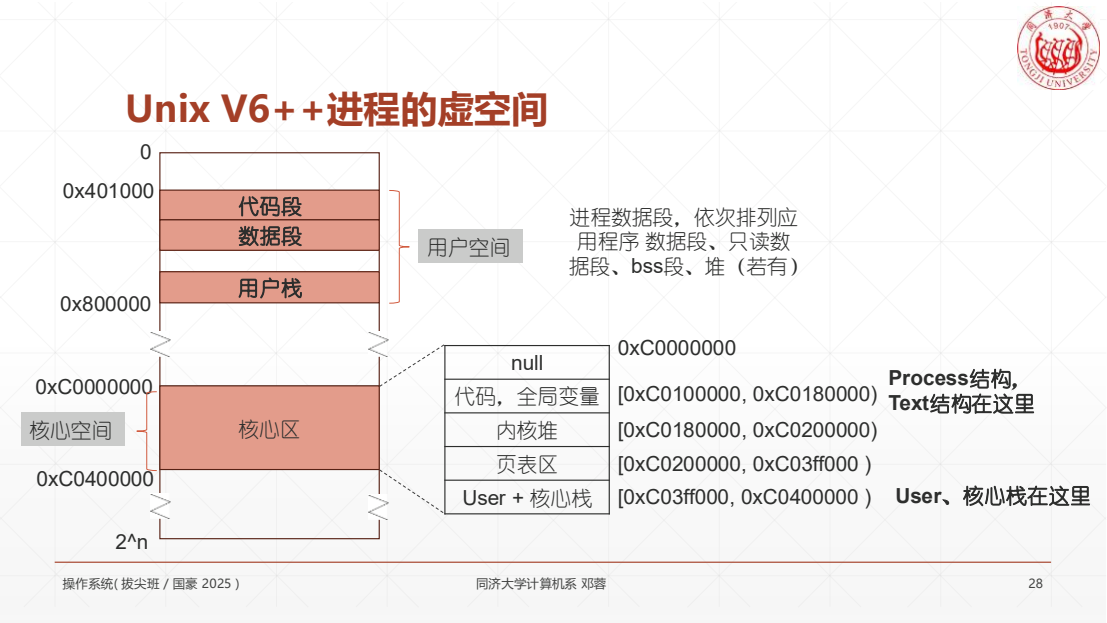


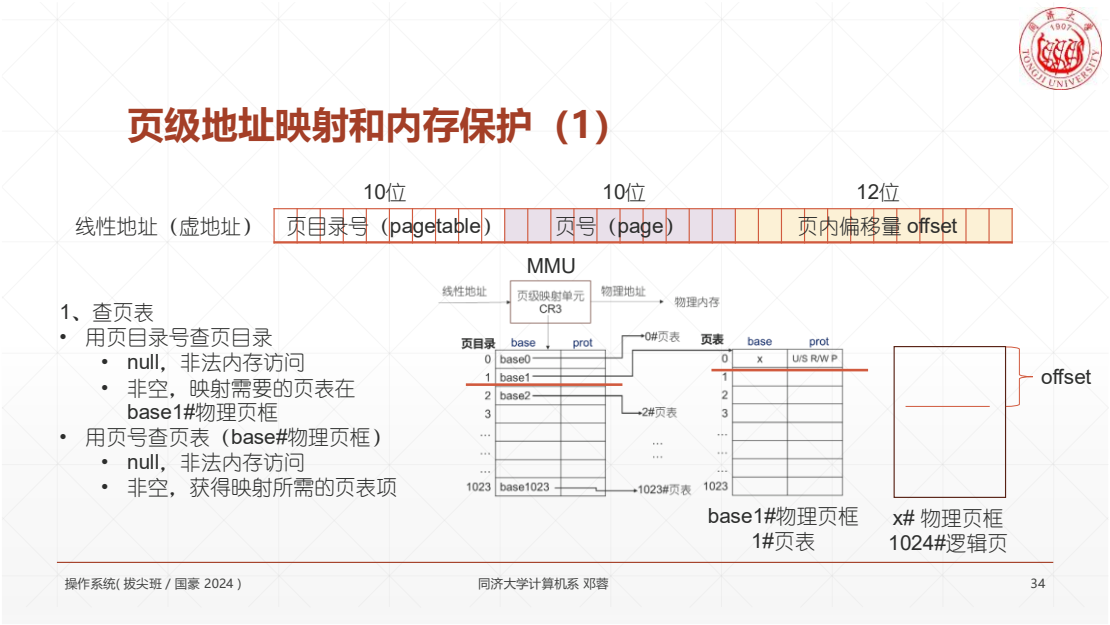
图 3、Unix V6++进程图像（虚空间）

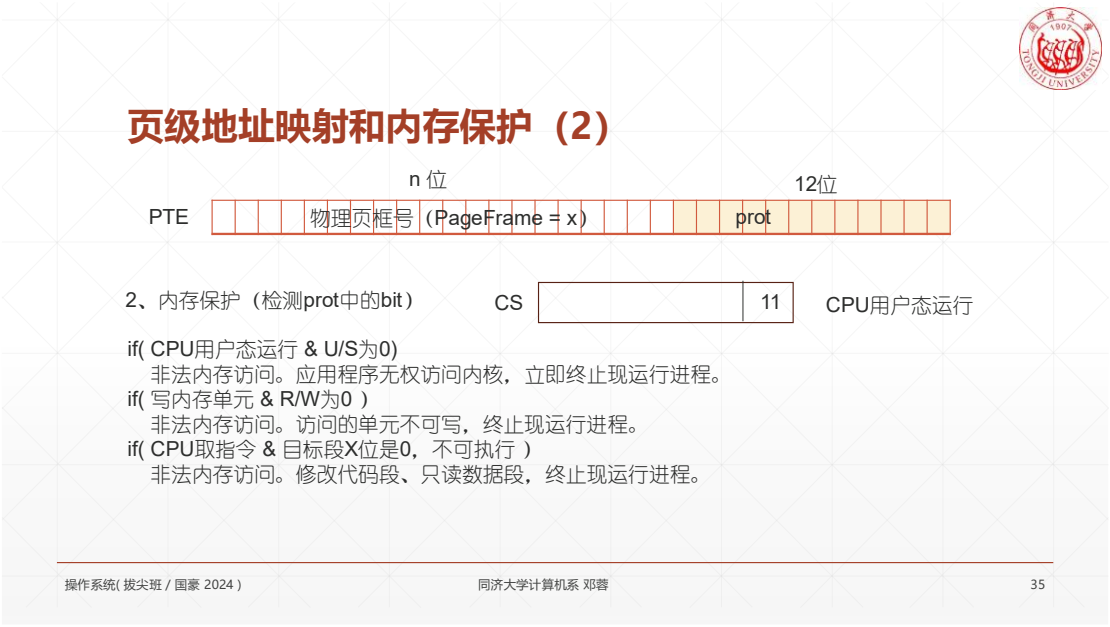
3、 相对虚实地址映射表

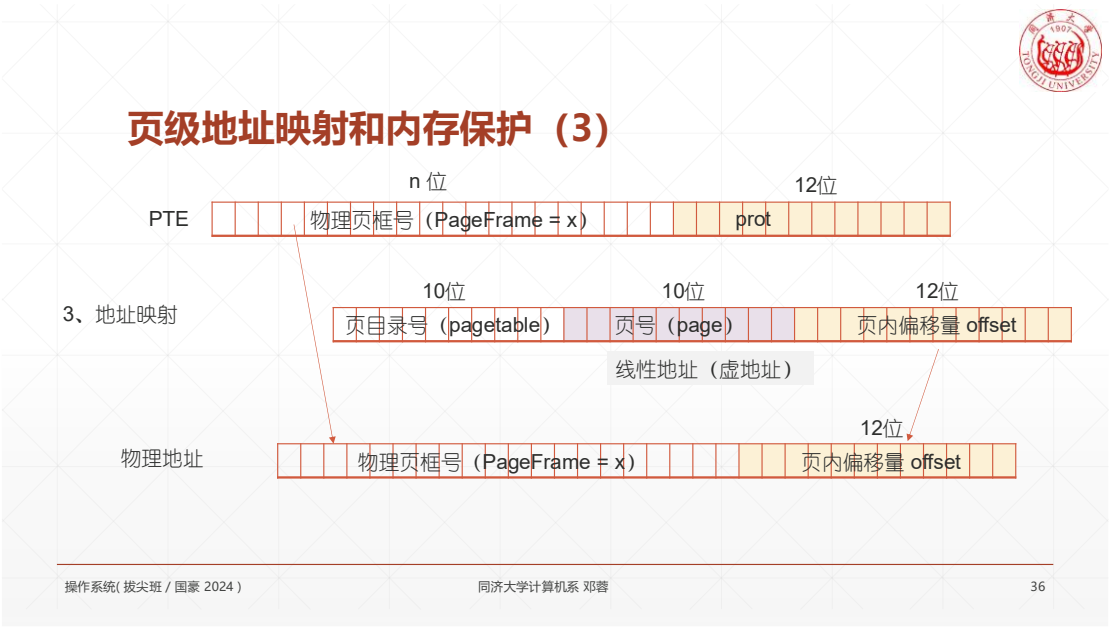
4、 MapToPageTable( )为新运行进程建立地址映射关系

- 以新进程相对表为模板，配合 p_addr 和 x_caddr，写系统用户页表

5、 运行时，CPU 执行单元给出逻辑地址，MMU 访问系统用户页表将其转换成物理地址。硬件逻辑如下：







6、Unix V6++使用的 MMU 是 i386，地址映射使用 2 级页表。
每次取指或者访问内存操作数，CPU 3 次访存：
用页目录号访问页目录，用页号访问页表，用物理地址访问物理页框。

作业 2

操作系统 2025~2026 期末考纲（下）

计算机拔尖班 2023，国豪 2023　　2025/12/31

二、 进程状态转换

1. Swtch( )
2. Sleep( )入睡、WakeUpAll( )唤醒
   要求：注释函数源代码，代码填空

三、 进程创建、终止、图像扩展、exec

1. Fork( )
2. Exit( )、Wait( )
   要求：会使用系统调用，知道系统调用执行过程，源代码不需要掌握
3. 扩展用户栈、malloc( ) 和 free( )
4. Exec
   要求：了解堆栈扩展过程。malloc、free 和 exec 不考

四、中断与调度

1、进程用户态运行时执行应用程序，使用用户栈
2、进程核心态运行时执行操作系统内核代码，使用核心栈

3、模式切换：

- 用户态→核心态，响应中断或执行系统调用（现运行进程执行系统调用也要仰仗中断机制）
- 核心态→用户态。中断返回（IRET）

为什么模式切换？

- 用户态→核心态，提权：让现运行进程拥有操作系统特权
  控制系统运行模式，访问系统资源。譬如：向外设发 IO 命令，访问 CPU 内部控制寄存器（比如页表寄存器 CR3），执行开关中断（CLI、STI）等特殊指令 ……
- 核心态→用户态，剥夺操作系统特权

模式切换的实现，关注中断入口程序

4、进程切换：
CPU 换现运行进程。

多道系统，进程切换的实现，关注 Swtch( )函数。

- 现场保护：核心态 ESP、EBP 存入现运行进程 user 结构，u_rsav 字段。这是 SaveU 宏在保护 Swtch( )栈帧。
- 0#进程执行 select( )函数，挑选一个合适的进程 PA。
- 恢复现场：
  - 从被选中进程 PA 的 user 结构，恢复 ESP、EBP 寄存器(Swtch 栈帧)，宏 RetU。
  - 为 CPU 建立 PA 进程的地址映射关系：函数 MapToPageTable()
- PA，Swtch( )函数返回、随后执行 Swtch 栈帧返回地址处保存的指令。。。得以从上次放

弃 CPU 的断点继续运行。

5、接下来，PA

- 执行系统调用下半段（如果是 0#进程，继续跑 Sched 函数执行换入、换出操作）。　　emm。。。。。上次 PA 放弃 CPU 一定是入睡
- 或，返回用户态执行应用程序。　　emm。。。。。上次 PA 放弃 CPU 是被剥夺（响应中断，唤醒了一个进程 或 PA 时间片到）

注：排除 PA 是 0#进程的情况。

6、没有模式切换，就没有进程切换。　　对

不全对，核心态线程入睡（譬如，0#进程）就没有模式切换。特例，记住上面的判断。

7、4 种中断：外设中断，时钟中断，异常，系统调用。

异常和信号处理不考

系统调用是应用程序调用内核子函数，函数调用方法与普通子程序调用很不一样。

表 3.1：一般程序调用与系统调用的区别

项目	一般子程序调用	系统调用
调用者	应用程序或内核中的任意子程序	应用程序中的某个子程序
被调用者	同一个程序中的另一个子程序	操作系统内部供应用程序调用的子程序
调用者和被调用者之间的关系	链接在同一个可执行文件中	分别链接在 2 个不同的可执行文件中。被调用者一定是操作系统内核
引发调用的事件	call 指令	INT 指令或 trap 指令
被调用者的栈帧位置	当前堆栈的栈顶	核心栈

Unix 系统时钟管理：　　时钟中断 + Sleep(seconds)系统调用

8、中断嵌套，中断优先级 和 进程优先级

中断嵌套，是指计算机执行低优先级中断处理程序时，响应高优先级中断请求，执行高优先级的中断处理程序。

执行中断处理程序的，是被中断的现运行进程。

中断优先级，中断处理程序的优先级：

- 时钟>键盘>磁盘；
- 系统调用和应用程序，中断优先级是 0。

进程优先级，系统调用的优先级：

- sched( )0#进程、最高优先权：-100 > 访问磁盘的系统调用：-50 > 访问键盘的系统调用：10 > Sleep(seconds)：90　　……
- 中断处理程序，优先级无穷大。体现在：中断处理程序没有全部执行完毕，CPU 不会执行任何其它任务。

9、中断返回前的例行调度

中断处理程序会改变调度子系统的状态：

- 外设中断，会唤醒等待 IO 完成的进程

- 时钟中断，会改变所有用户态进程（执行应用程序的进程）的优先数
- 系统调用，会改变进程的优先权

所以，Unix V6 在中断返回前安排了一个例行调度点：执行完所有中断处理程序（包括系统调用），返回用户态前夕，现运行进程将 CPU 让给优先级更高的就绪进程，自己被剥夺。

关键：RunRun 强迫调度标识。盯住中断入口函数和中断处理函数对 RunRun 标识的使用。

10、多道系统，并发的多个进程怎样轮流使用 CPU。

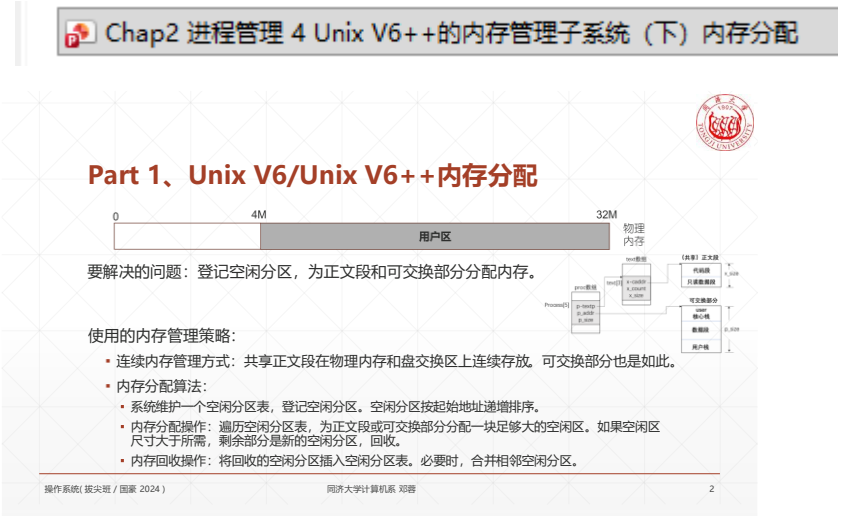
无论主动放弃 CPU，还是被剥夺，Unix 系统怎样保存现运行进程现场。

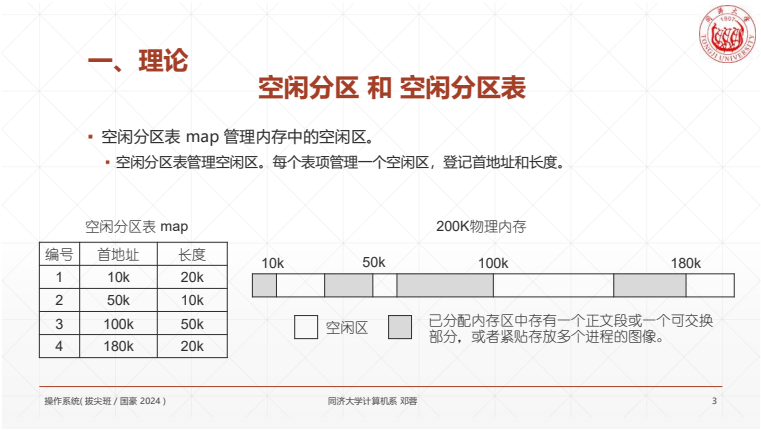
怎样保存、恢复应用程序的执行状态。
怎样保存、恢复系统调用的执行状态。 } 不必刻意复习，需要时，随意发挥

作业 8，作业 10

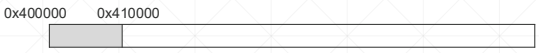
五、内存分配 & 盘交换区的使用

1、内存分配





情景分析，例题1



- T0时刻，用户区如图所示。注释：阴影部分已分配，除了PA正文段和可交换部分，还紧贴存放有其它进程的正文段和可交换部分。
- PA进程代码段3页，数据段1页，堆栈段1页；x_caddr=0x402000，p_addr=0x407000
- PA创建子进程PB。

（1）PB进程， x_caddr = （ 0x402000 ），p_addr = （ 0x410000 ）

2、盘交换区的使用

作业 10

必要时，参考这份 PPT，修了几页，容易读一些

Chap2 进程管理 12 使用盘交换区

1、系统初始化后，盘交换区的状态

- RunIn是0，RunOut是1。内存足够，0#进程等待换入任务:
  - **RunOut++，Sleep(&RunOut);**
- 在盘交换区出现第一个进程前，0#进程 SSLEEP。其间，
  - 进程切换时，0#进程执行 Select 选择新运行进程
  - 系统idle时，0#进程是现运行进程，等中断
    - 时钟中断，维护时钟
    - 其它会唤醒进程的中断，会唤醒睡眠进程。随后，0#进程选中新就绪进程，select返回后将CPU让给这个进程。回看Swtch、Select函数。
- 盘交换区出现第一个进程 PA 后，系统唤醒0#进程:
  - **Runout = 0**
  - **WakeUpAll(&Runout);**

盘交换区上的第一个进程是怎么来的？

2、0#进程首次工作

如果正好有SWAIT，对换成功后，0#进程 RunOut++；Sleep(&RunOut)。
否则，内存没空，盘交换区上有一个SRUN，0#进程 RunIn++；Sleep(&RunIn)。

```
Sched ( )
{
   goto loop;

Sloop:
   RunIn++，Sleep(&RunIn);

2 loop: // 换入
   寻找盘交换区p_time最大的就绪进程 PA，second是其驻留时长
   if（不存在）// 等待换入任务
      RunOut++；Sleep(&RunOut)； goto loop；
   else
      为PA分配内存；
      成功，goto found2
```

```
   不成功，选择换出进程PB、执行对换操作
3 • 有SWAIT，goto found1;
4 • 无，考虑换出 SRUN 或 SSLEEP，选p_time最大的
      (避免抖动，seconds>=3s & PB的 p_time>=2s)
      找到，goto found1;
      没找到，goto sloop；  // 入睡等待内存有空

3 found1：换出PB
   PB->p_flag &= ~Process::SLOAD;
   this->XSwap(PB, true, 0); // 释放其占用的内存空间
   goto loop; // 尝试换入PA

   found2：换入PA
      PPT 5、6 所示换入操作；  // 换出操作
      goto loop;
}
```


1、0#进程被唤醒后，goto loop
2、找到盘交换区上的就绪进程PA，为其分配内存，一定不成功
3、启动开销小的对换操作：换出内存中的进程PB（优选SWAIT），换入PA
• 成功。完美的布局！低优先级睡眠进程放盘交换区，就绪进程放内存
4、不成功，启动开销大的对换操作：对换内存中的就绪进程或高优先级进程
• 成功。内存告急，就绪、高优先级睡眠进程在轮流使用内存！
5、不成功，内存暂时无法容纳PA。0#进程 RunIn++；Sleep(&RunIn)等待可用的内存空间

3、现运行进程PB低优先权入睡前激活对换操作：

```
void Process::Sleep(unsigned long chan, int pri)
{
   if ( pri > 0 )
   {
      ……
      if ( procMgr.RunIn != 0 )
      {
         procMgr.RunIn = 0;
         procMgr.WakeUpAll((unsigned long)&procMgr.RunIn);
      }
      ……  Swtch( );
   }
}
```

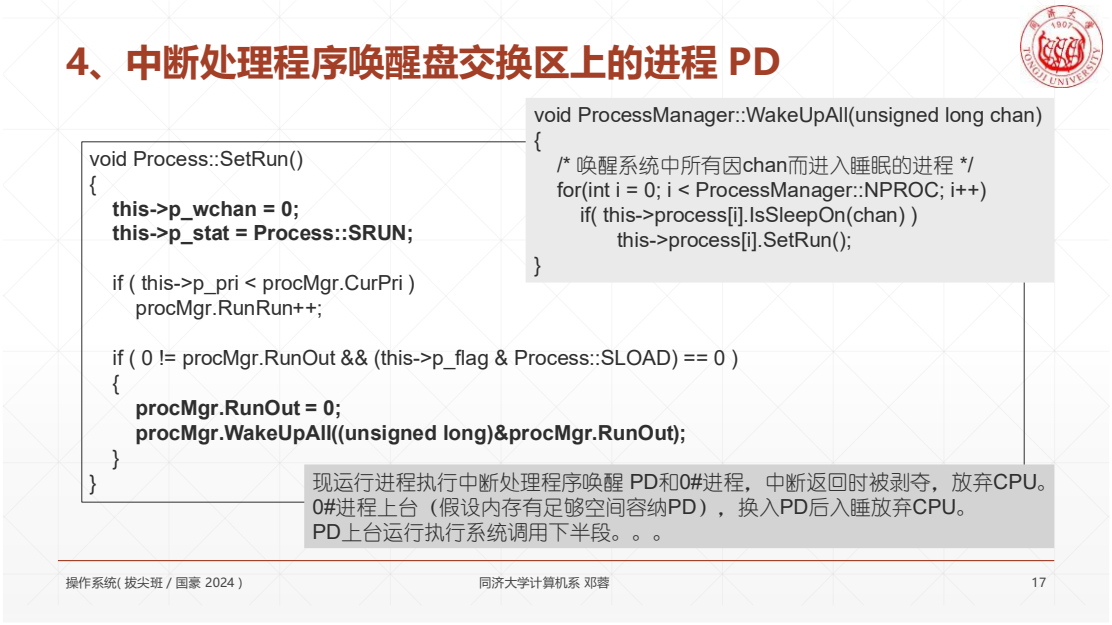
PB唤醒0#进程后放弃CPU。
0#进程上台，换出PB、换入PA （假设PB尺寸大于PA）。
0#进程入睡，PA上台运行。

```
Sched ( )
{
   goto loop;

Sloop:
   RunIn++，Sleep(&RunIn);

loop: // 换入
   寻找盘交换区p_time最大的就绪进程 PA，
其驻留时长
   if（不存在）// 等待换入任务
      RunOut++；Sleep(&RunOut)； go
   else
      为PA分配内存；
      成功，goto found2
```

4、中断处理程序唤醒盘交换区上的进程 PD



```
void ProcessManager::WakeUpAll(unsigned long chan)
{
    /* 唤醒系统中所有因chan而进入睡眠的进程 */
    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 )
    {
        procMgr.RunOut = 0;
        procMgr.WakeUpAll((unsigned long)&procMgr.RunOut);
    }
}
```

现运行进程执行中断处理程序唤醒 PD和0#进程，中断返回时被剥夺，放弃CPU。
0#进程上台（假设内存有足够空间容纳PD），换入PD后入睡放弃CPU。
PD上台运行执行系统调用下半段。。。

六、外设管理

1、硬盘 IO 性能优化

2、基于磁盘高速缓存的磁盘数据块读写

不要求掌握源代码，不要求掌握缓存队列操作细节

对照考纲，认真看例题习题

七、文件管理

1、文件系统静态结构

2、文件系统的使用

打开文件结构

目录搜索

系统调用 open，read，write，lseek，close 的执行过程

不要求掌握源代码

对照考纲，认真看例题习题