习题课

UNIX进程控制

方　钰

主要知识点：

1. UNIX进程的proc结构和user结构
2. 时钟中断与系统调用
3. UNIX的进程调度状态
4. UNIX的动态优先权调度算法
5. 主要的内核函数

主要知识点：

1. UNIX进程的proc结构和user结构
2. 时钟中断与系统调用
3. UNIX的进程调度状态
4. UNIX的动态优先权调度算法
5. 主要的内核函数

Process类

	名称	类型	含义
进程标识	p_uid	short	用户ID
	p_pid	int	进程标识数，进程编号
	p_ppid	int	父进程标识数
进程图象在内存中的位置信息	p_addr	unsigned long	ppda区在物理内存中的起始地址
	p_size	unsigned int	进程图象（除代码段以外部分）的长度，以字节单位
	p_textp	Text *	指向该进程所运行的代码段的描述符
进程调度相关信息	p_stat	ProcessState	进程当前的调度状态
	p_flag	int	进程标志位，可以将多个状态组合
	p_pri	int	进程优先数
	p_cpu	int	cpu值，用于计算p_pri
	p_nice	int	进程优先数微调参数
	p_time	int	进程在盘交换区上（或内存内）的驻留时间
	p_wchan	unsigned long	进程睡眠原因
信号与控制台终端	p_sig	int	进程信号
	p_ttyp	TTy*	进程tty结构地址

User类

	名称	类型	含义
进程的用户标识	u_uid	short	有效用户ID
	u_gid	short	有效组ID
	u_ruid	short	真实用户ID
	u_rgid	short	真实组ID
进程的时间相关	u_utime	int	进程用户态时间
	u_stime	int	进程核心态时间
	u_cutime	int	子进程用户态时间总和
	u_cstime	int	子进程核心态时间总和
现场保护相关	u_rsav[2]	unsigned long	用于保存esp与ebp指针
	u_ssav[2]	unsigned long	用于对esp和ebp指针的二次保护
内存管理相关	*u_procp	Process	指向该u结构对应的Process结构
	u_MemoryDescriptor	MemoryDescriptor	封装了进程的图象在内存中的位置、大小等信息
系统调用相关	EAX = 0	static const int	访问现场保护区中EAX寄存器的偏移量
	*u_ar0	unsigned int	指向核心栈现场保护区EAX寄存器存放的栈单元
	u_arg[5];	int	存放当前系统调用参数
	*u_dirp	char	系统调用参数（一般用于Pathname）的指针

ProcessManager类

名称	类型	含义
process[NPROC]	Process	进程基本控制块数组
text[NTEXT]	Text	代码段控制块数组
CurPri	int	现运行占用CPU时优先数
RunRun	int	强迫调度标志
RunIn	int	内存中无合适进程可以调出至盘交换区
RunOut	int	盘交换区中无进程可以调入内存
ExeCnt	int	同时进行图像改换的进程数
SwtchNum	int	系统中进程切换次数

主要知识点：

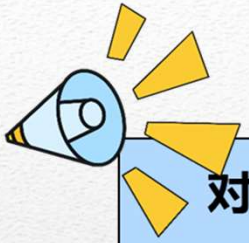
1. UNIX进程的proc结构和user结构
2. 时钟中断与系统调用
3. UNIX的进程调度状态
4. UNIX的动态优先权调度算法
5. 主要的内核函数

主要知识点：2. 时钟中断和系统调用

时钟中断完成的主要工作

时钟中断以每秒60次的频率定时自动发生

和进程是否发起IO操作无关



对p_cpu的处理

- 进程优先数的计算

每次心跳一次

- u_utime, u_stime的计数
- p_cpu
- lbolt的计数

简单，迅速完成

每1秒钟一次

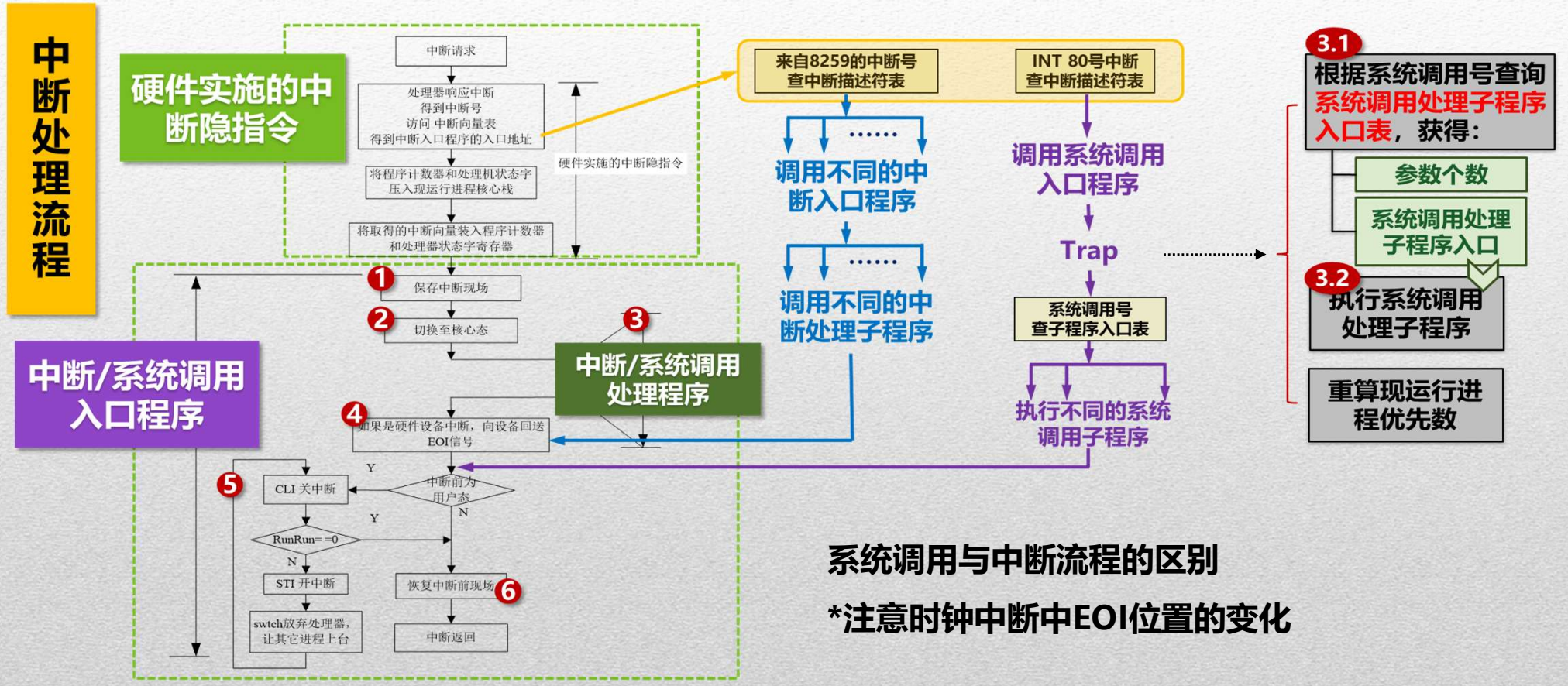
- 维护系统时间
- 修改所有进程的p_time
- 调整所有进程的p_cpu
- 重算部分进程的优先级
- 可能唤醒0#进程
- 重算当前进程的优先级

繁琐，耗时

- 先前态是用户态才做
- 提前开中断，EOI

主要知识点：2. 时钟中断和系统调用

系统调用的流程



系统调用与中断流程的区别

*注意时钟中断中EOI位置的变化

2. 关于时钟中断与系统调用

（1）下列关于系统调用的叙述中，正确的是（ C ）

Ⅰ. 在执行系统调用服务程序的过程中，CPU处于内核态

Ⅱ. 操作系统通过提供系统调用避免用户程序直接访问外设

Ⅲ. 不同的操作系统为应用程序提供了统一的系统调用接口

Ⅳ. 系统调用是操作系统内核为应用程序提供服务的接口

A. 仅Ⅰ、Ⅳ　　B. 仅Ⅱ、Ⅲ

C. 仅Ⅰ、Ⅱ、Ⅳ　　D. 仅Ⅰ、Ⅲ、Ⅳ

（2）执行系统调用的过程所包括的如下主要操作：

①返回用户态　　②执行陷入(trap)程序/指令

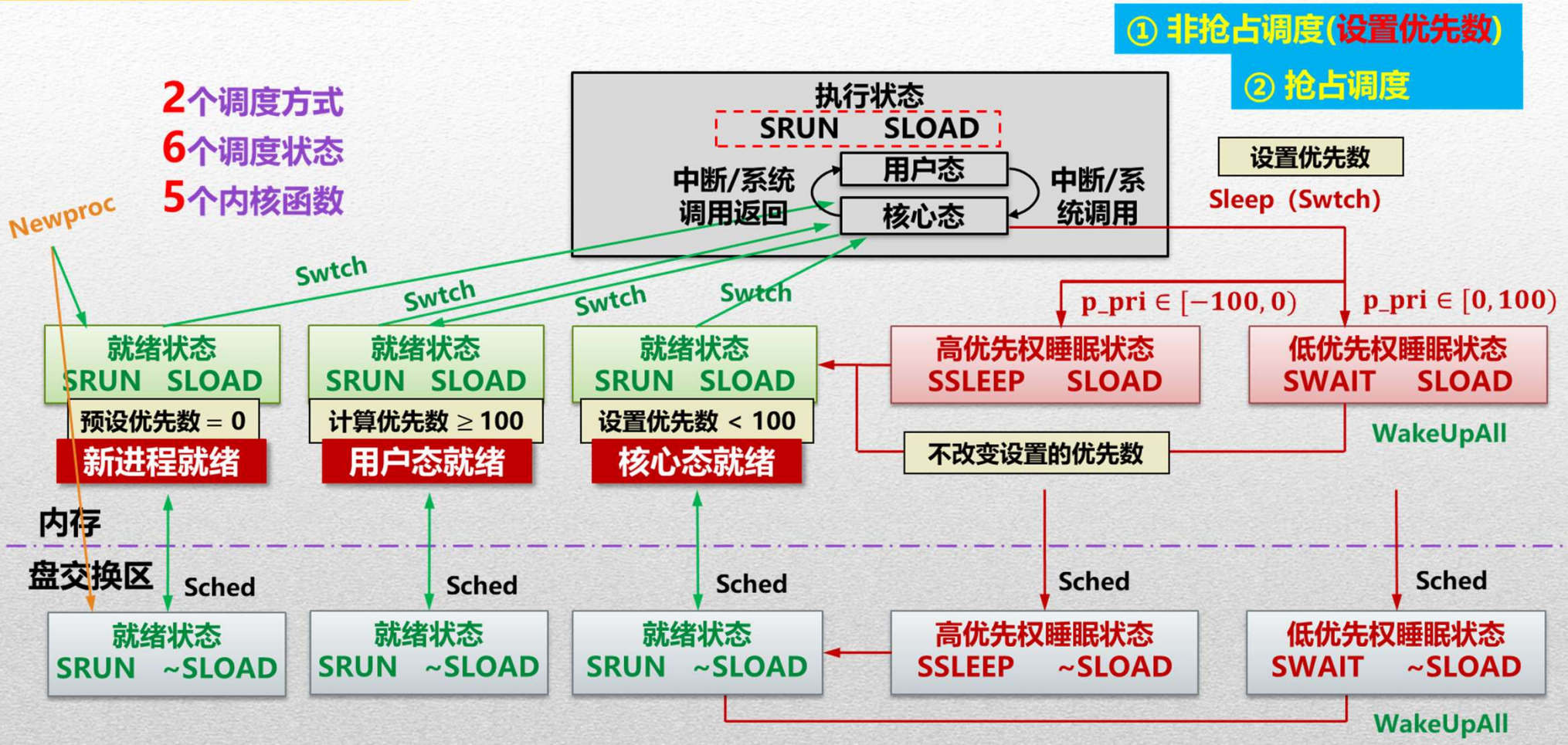
③传递系统调用参数　　④执行相应的服务程序

正确的执行顺序是：_____③→②→④→①_____。

主要知识点：

1. UNIX进程的proc结构和user结构
2. 时钟中断与系统调用
3. UNIX的进程调度状态
4. UNIX的动态优先权调度算法
5. 主要的内核函数

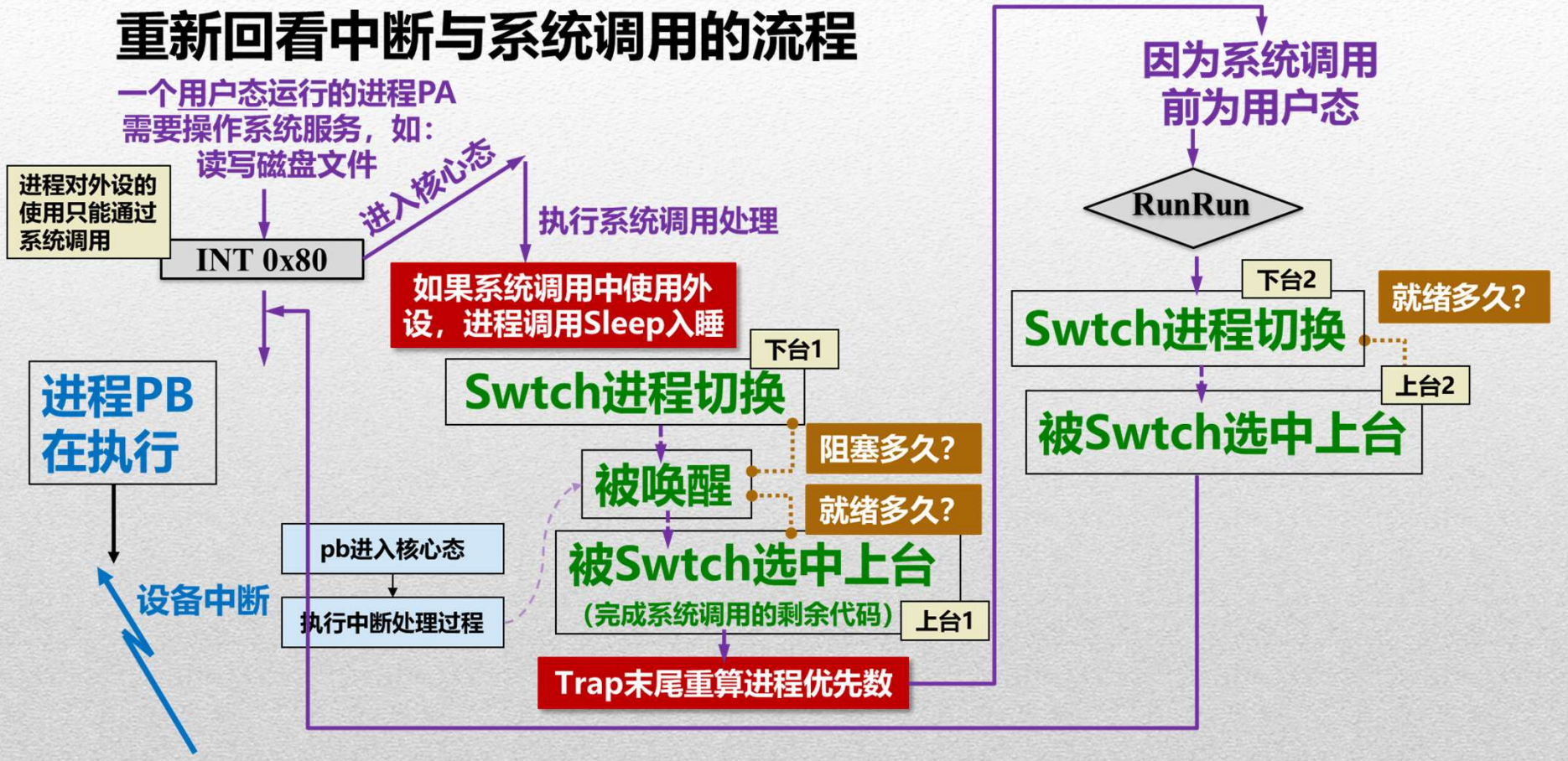
主要知识点：3. UNIX的进程调度状态



主要知识点：

3. UNIX的进程调度状态

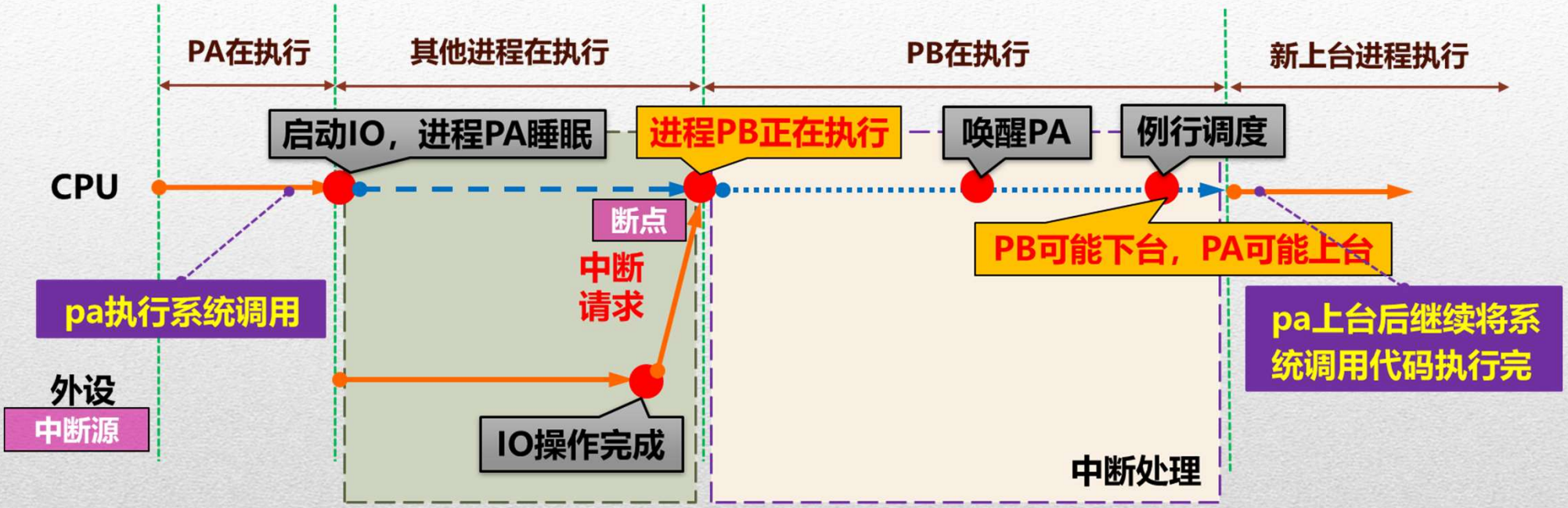
系统调用和中断的关系



主要知识点：

3. UNIX的进程调度状态

系统调用和中断的关系



1. 保证了CPU和设备之间的并行操作

2. 提供了进程执行内核代码的机会

3. 多道程序并发的硬件基础

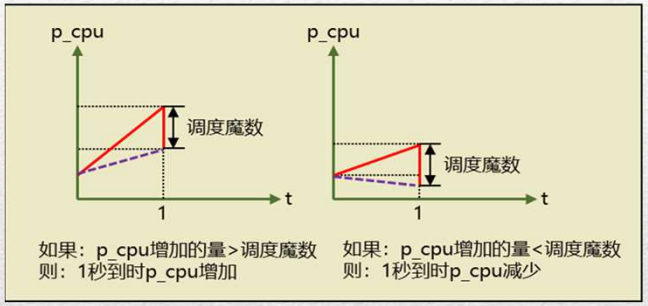
主要知识点：

1. UNIX进程的proc结构和user结构
2. 时钟中断与系统调用
3. UNIX的进程调度状态
4. UNIX的动态优先权调度算法
5. 主要的内核函数

主要知识点：

4. UNIX的动态优先权调度算法

计算优先数 $p\_pri = \min\{255, (p\_cpu / 16 + PUSER + p\_nice)\}$



1. 连续占用处理机较长时间的进程：
   p_cpu ↗，优先数↗，优先权↘
2. 较长时间内未使用处理机的进程：
   p_cpu ↘，优先数↘，优先权↗；
3. 虽频繁使用处理机，但每次时间很短的进程：
   p_cpu ↘，优先数↘，优先权↗。

根据睡眠原因

系统调用	系统调用中，如果进程入睡，设置其优先数	3. 在系统调用末尾，重算现运行进程的优先数	2. 每一秒，重新计算现运行进程的优先级	1. 每一秒，对所有优先数 > PUSER的进程，重新计算优先数p_pri	时钟中断

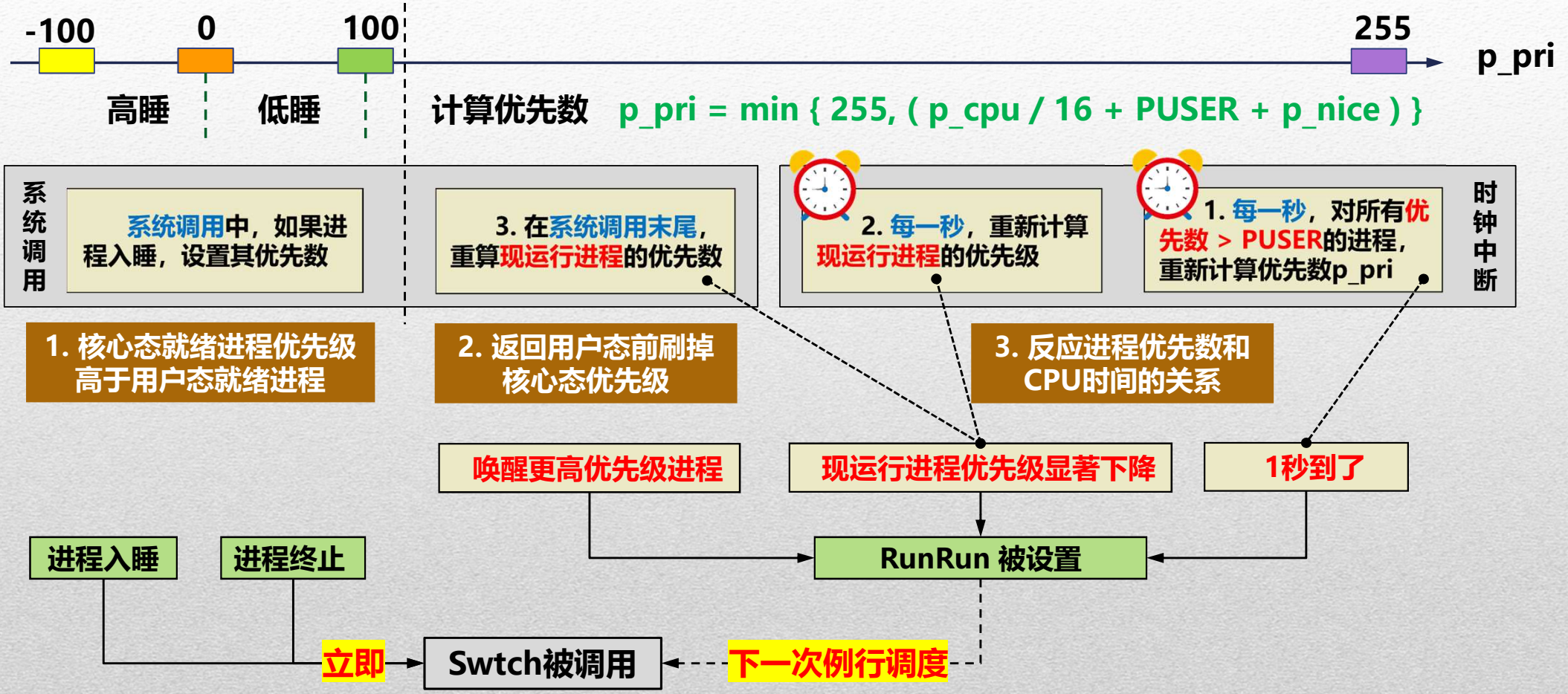
1. 核心态就绪进程优先级高于用户态就绪进程

2. 返回用户态前刷掉核心态优先级

3. 反应进程优先数和CPU时间的关系

主要知识点：

4. UNIX的动态优先权调度算法



1. 请回答下列问题：

（1）父进程创建子进程时，如果内存空间不足，子进程的图像将创建在哪里？此时子进程的状态是什么（p_stat和p_flag）？何时子进程的图像被搬入内存？子进程第一次被调度上台，从Swtch返回后，执行哪一条指令？

答：父进程创建子进程时，如果内存空间不足，子进程的图像将被创建在盘交换区上。此时，子进程的状态为： p_stat=SRUN，p_flag=~SLOAD|SSWAP。

子进程作为一个在盘交换区上就绪的进程，未来会在某一个时刻被0#进程将图像调入内存。

子进程未来如果在Swtch中被0#进程选中，从Swtch返回后执行的第一条指令是Newproc的下一条指令。

1. 请回答下列问题：

（2）当以下标志出现时，UNIX会产生什么动作?

答：RunRun==1：抢占调度标志，表示现运行进程可能已经不适合继续占用处理机（优先级下降、1秒计时到、有更高优先级进程就绪）。在随后的一次例行调度中（一次先前态为用户态进程的中断返回或系统调用返回前），将调用Swtch，实施一次进程切换调度。

RunIn==1：　表示盘交换区上有进程需要进入内存，但内存无足够的空间可供调入，且没有可供换出的进程。此时，0#进程因为RunIn进入睡眠状态。未来，如果内存出现可供换出的进程（有进程进入低睡状态或1秒计时到），0#进程被唤醒，重新尝试寻找可供换出的进程图像。

1. 请回答下列问题：

（2）当以下标志出现时，UNIX会产生什么动作?

答：RunOut==1：表示盘交换区上没有就绪状态的进程需要进入内存。此时，0#进程因为RunOut进入睡眠状态。未来，如果有一个盘交换区上睡眠的进程被唤醒，将同时唤醒0#进程，醒来的0#进程上台后，将该进程的图像换入内存。

1. 请回答下列问题：

（3）UNIX中存在核心态进程从运行状态转变为就绪状态的情况吗？

UNIX所有的进程调度状态变化都发生在核心态。

（4）UNIX中抢占调度和非抢占调度分别在什么情况下发生？

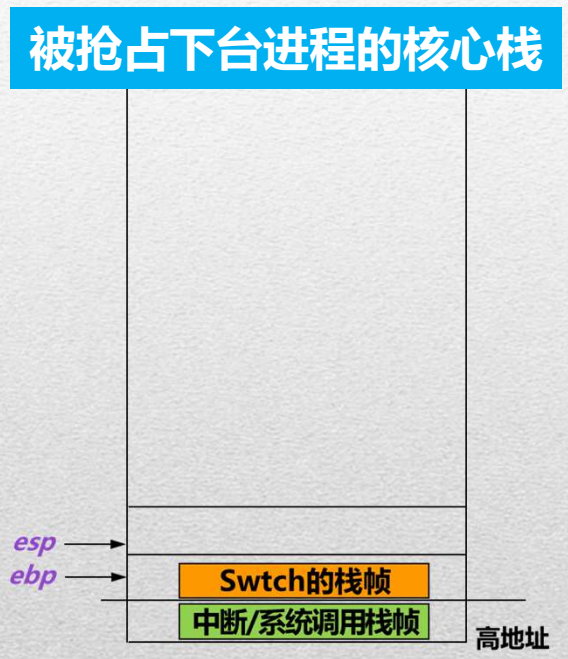
非抢占调度：现运行进程入睡或终止，执行Swtch。
抢占调度：现运行进程先前态为用户态时响应中断或执行系统调用，即将返回用户态时，若RunRun被设置，则执行Swtch。

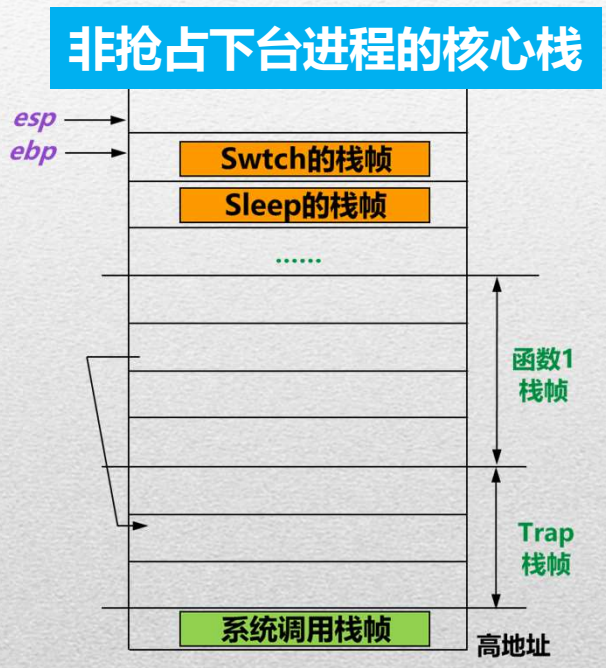
（5）由于抢占调度和非抢占调度下台的进程，它们的优先级有什么不同？？

非抢占调度下台的进程优先级高于抢占调度下台的进程。
前者是由设置获得的优先数<100。后者是由计算获得的优先数>=100。

1. 请回答下列问题：

（6）由于抢占调度和非抢占调度下台的进程，它们的核心栈有什么不同？





（7）抢占和非抢占下台的进程，它们重新上台后执行的操作有什么不同？

2. 情景分析题

假设某UNIX系统中，当前时刻t_0的进程状态如下表所示。且内存空间已满。

序号	占用空间	优先数	状态	位置	年龄
0#	-	-100	高睡 RunOut	SLOAD	-
p1	40K	105	就绪	SLOAD	2
p2	60K	110	执行	SLOAD	2
p3	60K	10	低睡	~SLOAD	3

请尽量详细地分析以下时刻系统中与进程调度相关的行为：

2. 情景分析题

序号	占用空间	优先数	状态	位置	年龄
0#	-	-100	高睡 RunOut	SLOAD	-
p1	40K	105	就绪	SLOAD	2
p2	60K	110	执行	SLOAD	2
p3	60K	10	低睡	~SLOAD	3

（1） t_0时刻现运行进程p2执行read系统调用：

进程p2由于执行磁盘I/O，调用内核函数Sleep，进入高睡状态，放弃处理器。Sleep中调用Swtch，选中内存中的就绪进程p1，使其占用处理器继续执行。

序号	占用空间	优先数	状态	位置	年龄
0#	-	-100	高睡 RunOut	SLOAD	-
p1	40K	105	执行	SLOAD	2
p2	60K	-50	高睡	SLOAD	2
p3	60K	10	低睡	~SLOAD	3

序号	占用空间	优先数	状态	位置	年龄
0#	-	-100	高睡 RunOut	SLOAD	-
p1	40K	105	执行	SLOAD	2
p2	60K	-50	高睡	SLOAD	2
p3	60K	10	低睡	~SLOAD	3

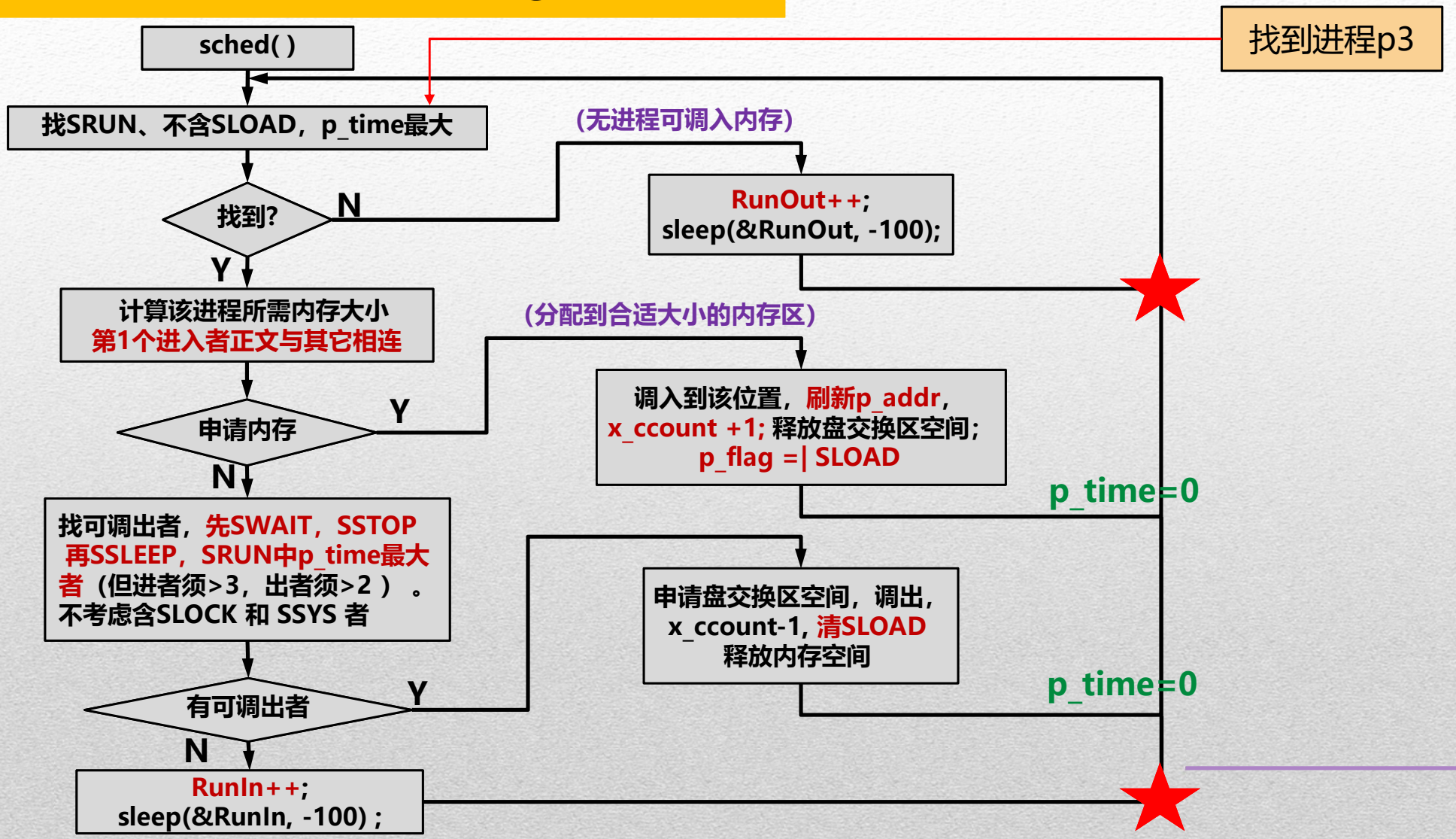
2. 情景分析题

（2） 1秒后，p1在用户态执行，p3的I/O完成：

进程p1执行中断处理程序，唤醒p3，因为p3的图像在盘交换区，且0#进程因为RunOut睡眠，则唤醒0#。 RunRun被设置。中断返回的例行调度中，0#上台。

序号	占用空间	优先数	状态	位置	年龄
0#	-	-100	执行	SLOAD	-
p1	40K	107	就绪	SLOAD	3
p2	60K	-50	高睡	SLOAD	3
p3	60K	10	就绪	~SLOAD	4

0#进程执行ProcessManager::Sched



序号	占用空间	优先数	状态	位置	年龄
0#	-	-100	执行	SLOAD	-
p1	40K	108	就绪	SLOAD	3
p2	60K	-50	高睡	SLOAD	3
p3	60K	10	就绪	~SLOAD	4

2. 情景分析题

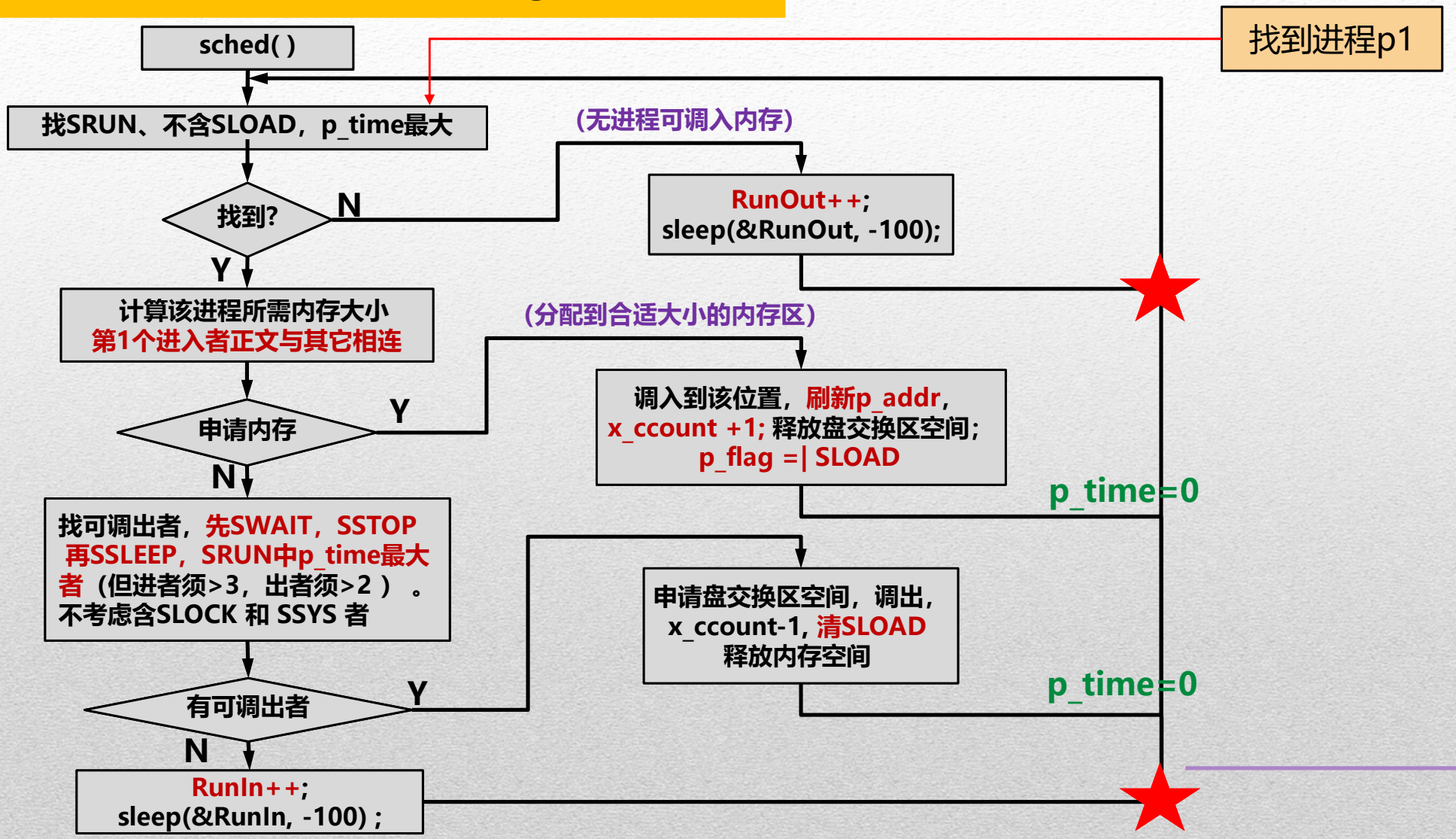
（2） 1秒后，p3进程的I/O完成：

0#选择p1换出，不够；

0#再选择p2换出，p3换入。

序号	占用空间	优先数	状态	位置	年龄
0#	-	-100	执行	SLOAD	-
p1	40K	108	就绪	~SLOAD	0
p2	60K	-50	高睡	~SLOAD	0
p3	60K	10	就绪	SLOAD	0

0#进程执行ProcessManager::Sched



序号	占用空间	优先数	状态	位置	年龄
0#	-	-100	执行	SLOAD	-
p1	40K	108	就绪	~SLOAD	0
p2	60K	-50	高睡	~SLOAD	0
p3	60K	10	就绪	SLOAD	0

2. 情景分析题

（2） 1秒后，p3进程的I/O完成：

0#找到p1，换入；

序号	占用空间	优先数	状态	位置	年龄
0#	-	-100	执行	SLOAD	-
p1	40K	108	就绪	SLOAD	0
p2	60K	-50	高睡	~SLOAD	0
p3	60K	10	就绪	SLOAD	0

序号	占用空间	优先数	状态	位置	年龄
0#	-	-100	执行	SLOAD	-
p1	40K	108	就绪	SLOAD	0
p2	60K	-50	高睡	~SLOAD	0
p3	60K	10	就绪	SLOAD	0

2. 情景分析题

（2） 1秒后，p3进程的I/O完成：

0#找不到磁盘的就绪进程，睡觉。p3上台。

序号	占用空间	优先数	状态	位置	年龄
0#	-	-100	高睡 RunOut	SLOAD	-
p1	40K	108	就绪	SLOAD	0
p2	60K	-50	高睡	~SLOAD	0
p3	60K	10	执行	SLOAD	0

3. 代码分析题

```
#include  <stdio.h>
#include <sys.h>
main( )
{
  int i,j;
  if(fork())
  {
            i=wait(&j);
            printf("It is parent process. \n");
            printf("The finished child process is %d. \n", i);
            printf("The exit status is %d. \n", j);
  }
  else
  {
            printf("It is child process. \n");
            exit(1);
  }
}
```

假设系统中只有上述代码运行，试回答下列问题：

（1）请写出代码的输出结果（假设父进程的ID号为500，子进程的ID号为505）。

It is child process.
It is parent process.
The finished child process is 505.
The exit status is 1.

（2）终止子进程的PCB何时回收？由哪个进程回收?

子进程终止时，唤醒父进程，由父进程回收。

3. 代码分析题

```
#include  <stdio.h>
#include <sys.h>
main( )
{
  int i,j;
  if(fork())
  {
            sleep(6);
            printf("It is parent process. \n");
  }
  else
  {
            printf("It is child process. \n");
            exit(1);
  }
}
```

假设系统中只有上述代码运行，试回答下列问题：

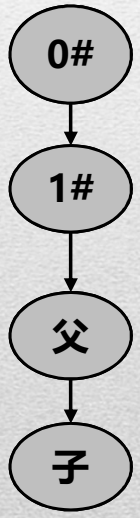
（1）终止子进程的PCB何时回收？由哪个进程回收？。

3. 代码分析题

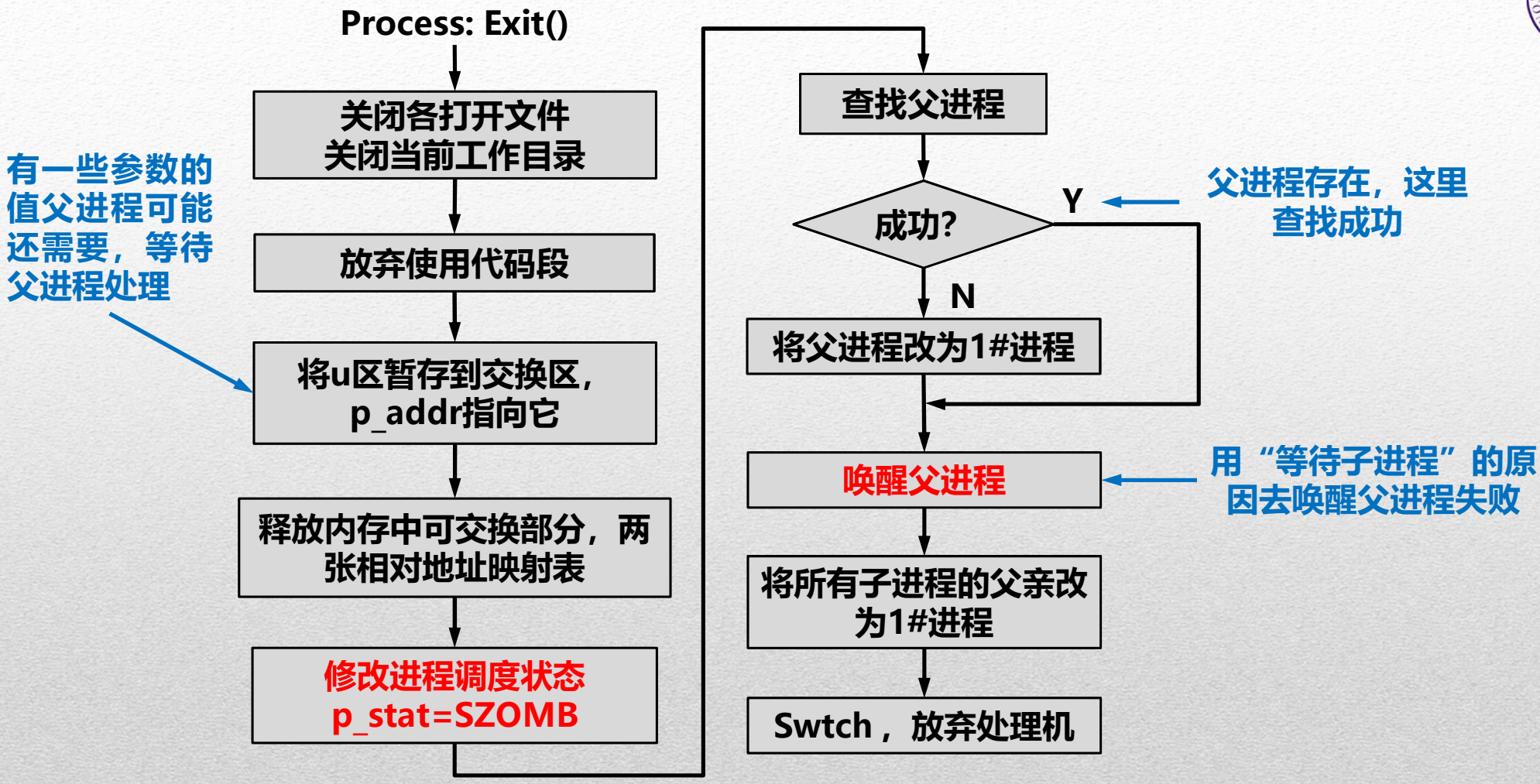
```
#include  <stdio.h>
#include <sys.h>
main( )
{
   int i,j;
   if(fork())
   {
                sleep(6);
                printf("It is parent process. \n");
   }
   else
   {
                printf("It is child process. \n");
                exit(1);
   }
}
```

假设系统中只有上述代码运行，试回答下列问题：

（1）终止子进程的PCB何时回收？由哪个进程回收？。



如果子进程先终止（子进程终止时父进程在睡觉）：

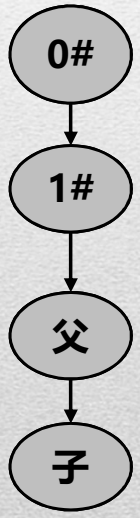

Process: Exit()
关闭各打开文件
关闭当前工作目录
放弃使用代码段
将u区暂存到交换区，
p_addr指向它
释放内存中可交换部分，两
张相对地址映射表
修改进程调度状态
p_stat=SZOMB
有一些参数的值父进程可能还需要，等待父进程处理
查找父进程
成功?
Y
N
父进程存在，这里查找成功
将父进程改为1#进程
唤醒父进程
用“等待子进程”的原因去唤醒父进程失败
将所有子进程的父亲改为1#进程
Swtch，放弃处理机

3. 代码分析题

```
#include  <stdio.h>
#include <sys.h>
main( )
{
  int i,j;
  if(fork())
  {
            sleep(6);
            printf("It is parent process. \n");
  }
  else
  {
            printf("It is child process. \n");
            exit(1);
  }
}
```

假设系统中只有上述代码运行，试回答下列问题：

（1）终止子进程的PCB何时回收？由哪个进程回收？。



如果子进程先终止（子进程终止时父进程在睡觉）：

父进程终止时，将子进程转交1#，并唤醒1#，子进程PCB由1#回收。

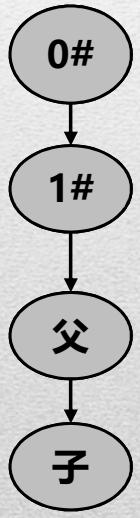
3. 代码分析题

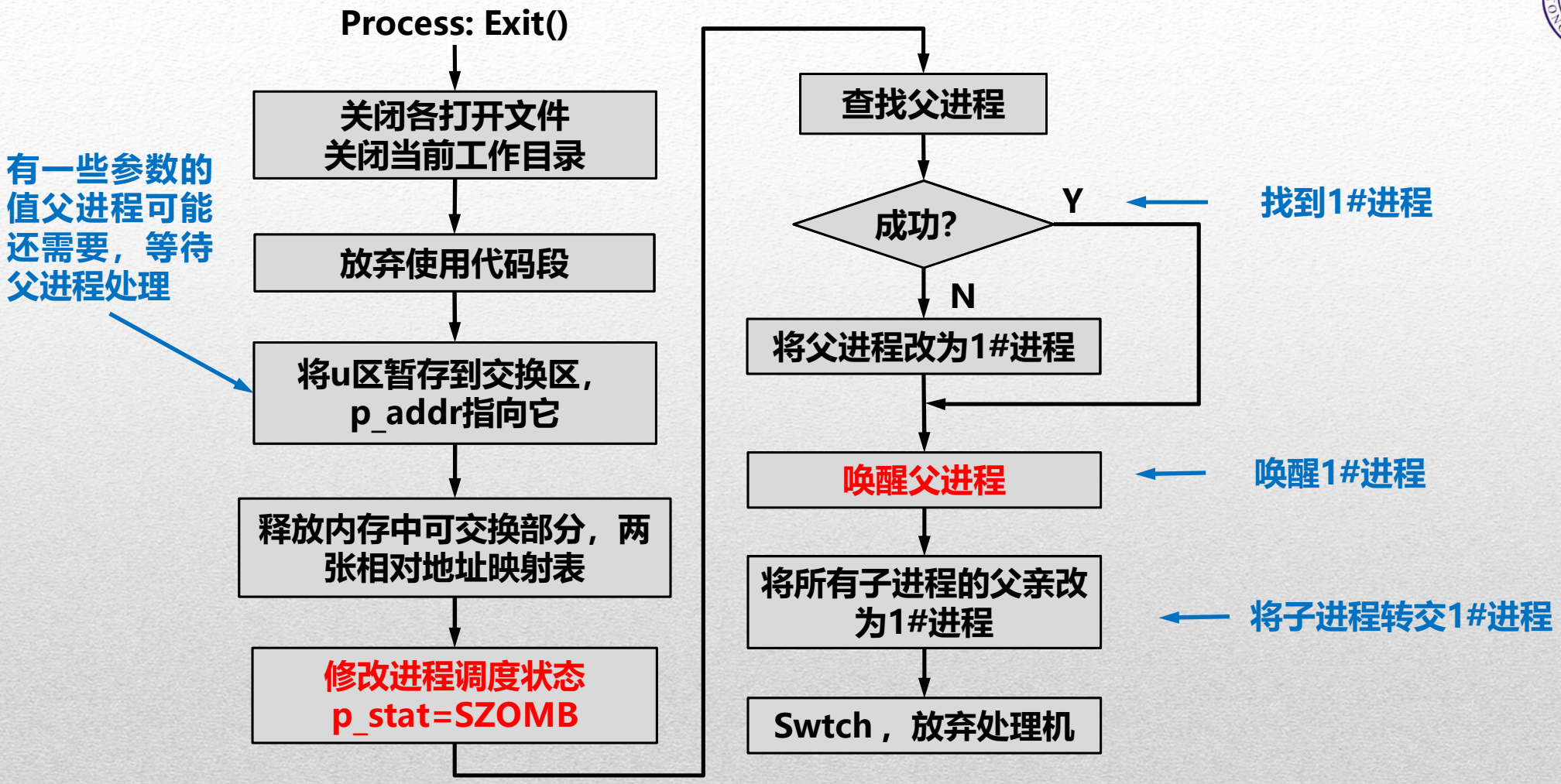
```
#include  <stdio.h>
#include <sys.h>
main( )
{
   int i,j;
   if(fork())
   {
                sleep(6);
                printf("It is parent process. \n");
   }
   else
   {
                printf("It is child process. \n");
                exit(1);
   }
}
```

假设系统中只有上述代码运行，试回答下列问题：

（1）终止子进程的PCB何时回收？由哪个进程回收？。

如果父进程先终止：



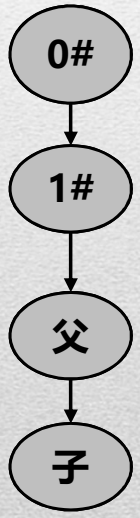

Process: Exit()
关闭各打开文件
关闭当前工作目录
放弃使用代码段
将u区暂存到交换区，
p_addr指向它
释放内存中可交换部分，两
张相对地址映射表
修改进程调度状态
p_stat=SZOMB
查找父进程
成功?
Y
找到1#进程
N
将父进程改为1#进程
唤醒父进程
唤醒1#进程
将所有子进程的父亲改
为1#进程
将子进程转交1#进程
Swtch，放弃处理机
有一些参数的
值父进程可能
还需要，等待
父进程处理

3. 代码分析题

```
#include  <stdio.h>
#include <sys.h>
main( )
{
   int i,j;
   if(fork())
   {
                sleep(6);
                printf("It is parent process. \n");
   }
   else
   {
                printf("It is child process. \n");
                exit(1);
   }
}
```

假设系统中只有上述代码运行，试回答下列问题：

（1）终止子进程的PCB何时回收？由哪个进程回收？。



如果父进程先终止：

父进程终止时，将子进程交给1#进程；并唤醒1#进程；

1#进程上台后，回收父进程PCB。

子进程终止时，唤醒1#进程，由1#进程回收PCB。