第四章

进程管理

主要知识点

1. UNIX进程的Process类，User类和ProcessManager类
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Process类，User类和ProcessManager类
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

UNIX V6++中与进程管理相关的类 - Process

	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

UNIX V6++中与进程管理相关的类 - Process

```
void SetRun();                                  /* 唤醒当前进程，转入就绪状态 */
bool IsSleepOn(unsigned long chan);             /* 检查当前进程睡眠原因是否为chan */
void Sleep(unsigned long chan, int pri);        /* 使当前进程转入睡眠状态 */
void Exit();                                    /* Exit()系统调用处理过程 */
void Clone(Process& proc);                      /* 除p_pid之外子进程拷贝父进程Process结构 */
```

和单个进程的调度控制相关

```
void SetPri();                          /* 根据占用CPU时间计算当前进程优先数 */
void Nice();                            /* 用户设置计算进程优先数的偏置值 */
```

和进程优先数的计算相关

```
void Expand(unsigned int newSize);      /* 改变进程占用的内存大小 */
void SStack();                          /* 堆栈溢出时，自动扩展堆栈 */
void SBreak();                          /* brk()系统调用处理过程 */
```

与进程图像的改变相关

UNIX V6++中与进程管理相关的类 - User

	名称	类型	含义
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
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

UNIX V6++中与进程管理相关的类 - ProcessManager

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

```
Process* Select();                          /* 选出最适合上台运行的进程 */
int Swtch();                                /* 进程的切换调度 */
void WakeUpAll(unsigned long chan);         /* 唤醒系统中所有因chan而进入睡眠的进程 */
void Wait();                                /* 父进程等待子进程结束的Wait()系统调用 */
```

```
int NewProc(); /* 用于生成当前正在运行进程的拷贝 */
void Fork();   /* 进程创建Fork()系统调用 */
```

```
void Sched();  /* 进程图像内存和交换区之间的传送 */
void XSwap(Process* pProcess, bool bFreeMemory, int size); /* 将进程从内存换出至交换区上 */
```

主要知识点

1. UNIX进程的Process类，User类和ProcessManager类
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

时钟中断与系统调用

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
- 两次计算优先数

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

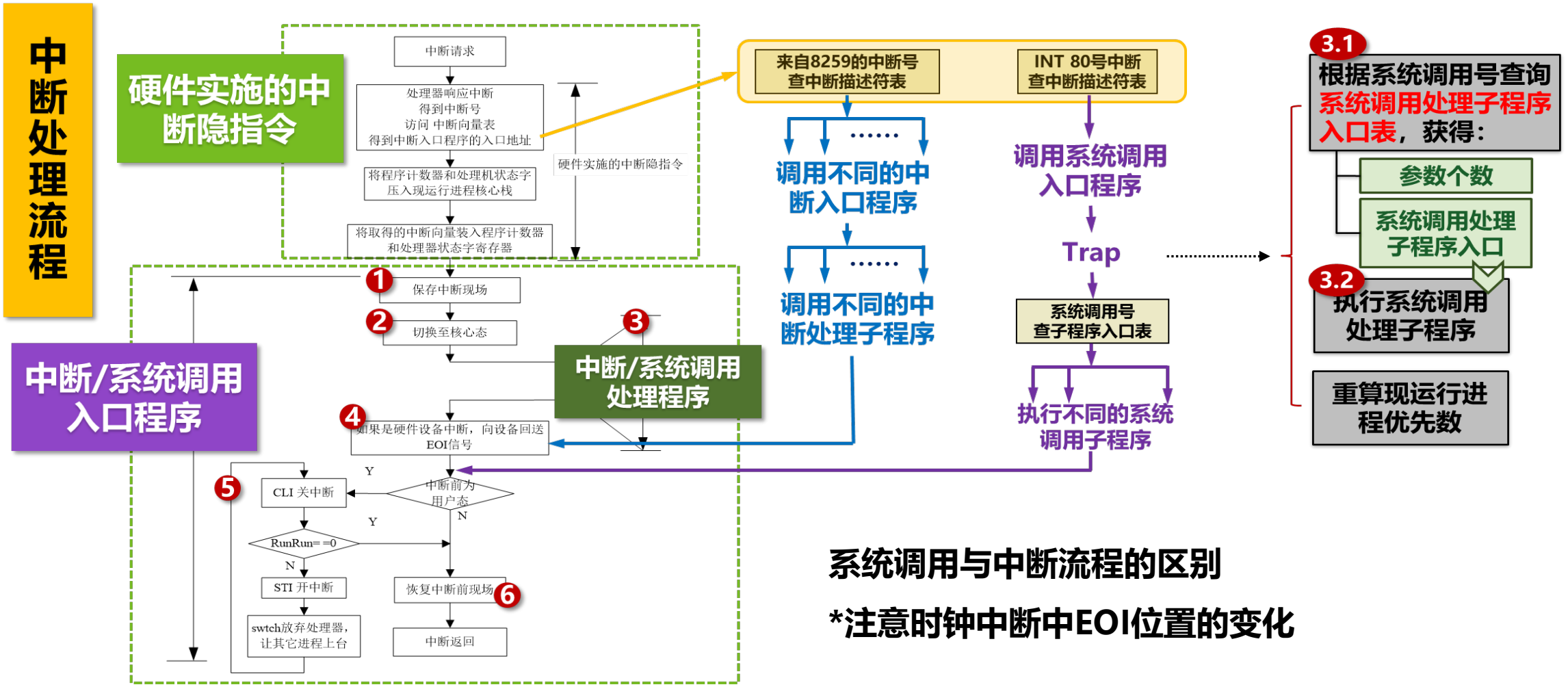
每1秒钟一次

- 维护系统时间
- 唤醒所有延时睡眠的进程
- 修改所有进程的p_time
- 调整所有进程的p_cpu
- 重算用户态就绪进程的优先级
- 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

时钟中断与系统调用



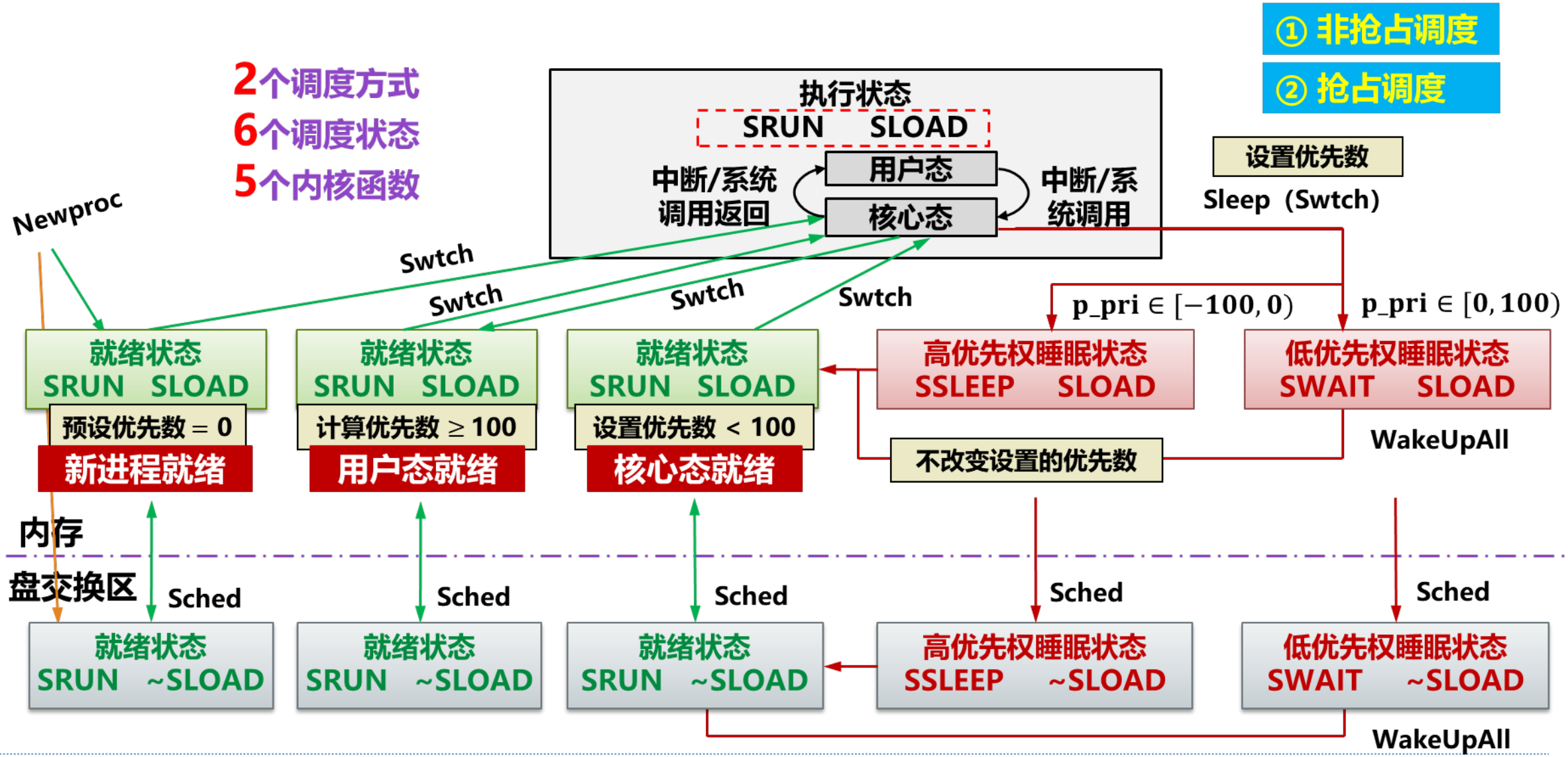
系统调用与中断流程的区别

***注意时钟中断中EOI位置的变化**

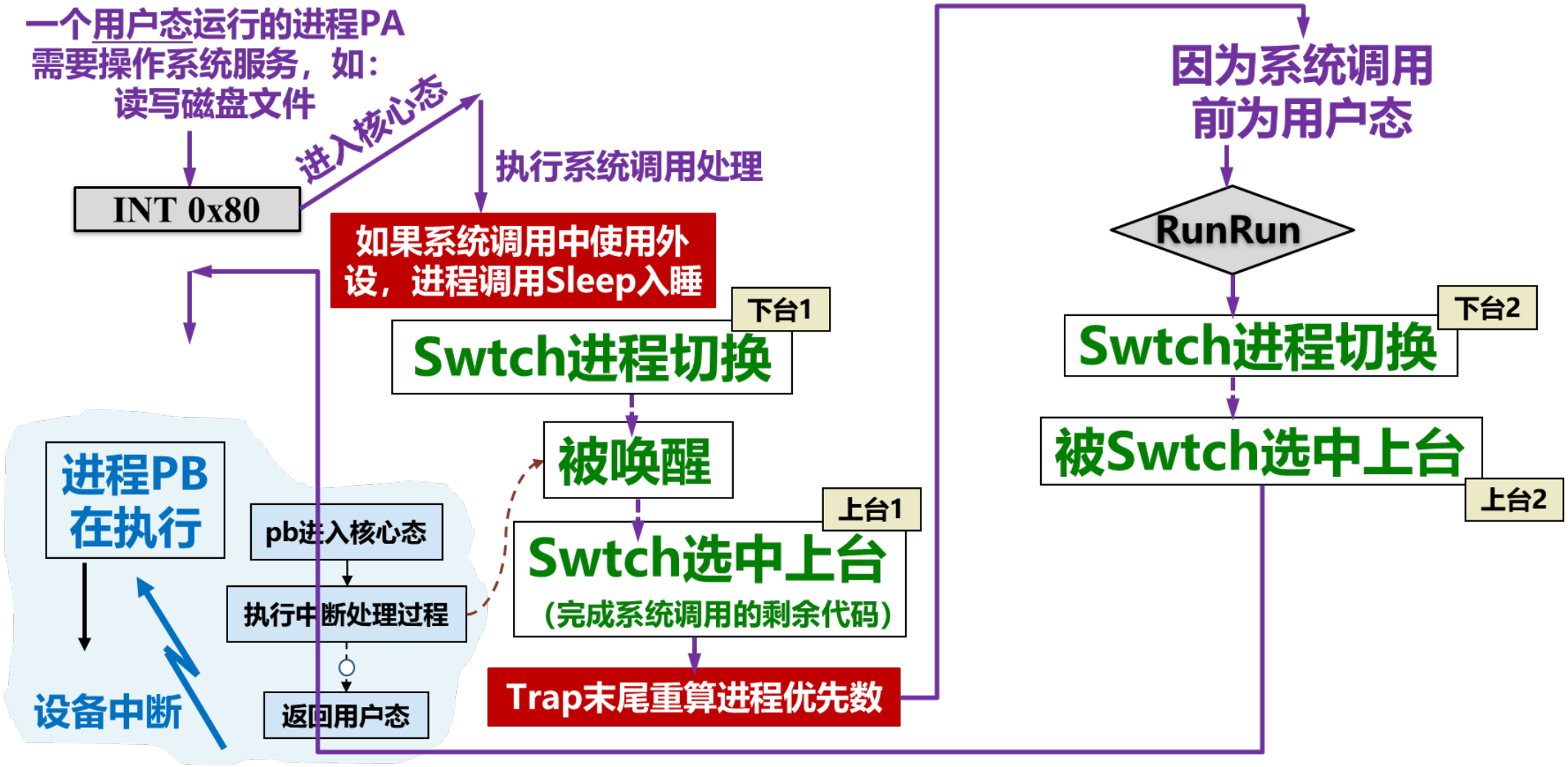
主要知识点

1. UNIX进程的Process类，User类和ProcessManager类
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

UNIX进程调度状态


① 非抢占调度
② 抢占调度
2个调度方式
6个调度状态
5个内核函数
执行状态
SRUN SLOAD
用户态
核心态
中断/系统调用返回
中断/系统调用
设置优先数
Sleep（Swtch）
Newproc
Swtch
Swtch
Swtch
Swtch
p_pri ∈ [−100, 0)
p_pri ∈ [0, 100)
就绪状态 SRUN SLOAD
预设优先数 = 0
新进程就绪
就绪状态 SRUN SLOAD
计算优先数 ≥ 100
用户态就绪
就绪状态 SRUN SLOAD
设置优先数 < 100
核心态就绪
高优先权睡眠状态 SSLEEP SLOAD
低优先权睡眠状态 SWAIT SLOAD
WakeUpAll
不改变设置的优先数
内存
盘交换区
Sched
Sched
Sched
Sched
Sched
就绪状态 SRUN ~SLOAD
就绪状态 SRUN ~SLOAD
就绪状态 SRUN ~SLOAD
高优先权睡眠状态 SSLEEP ~SLOAD
低优先权睡眠状态 SWAIT ~SLOAD
WakeUpAll

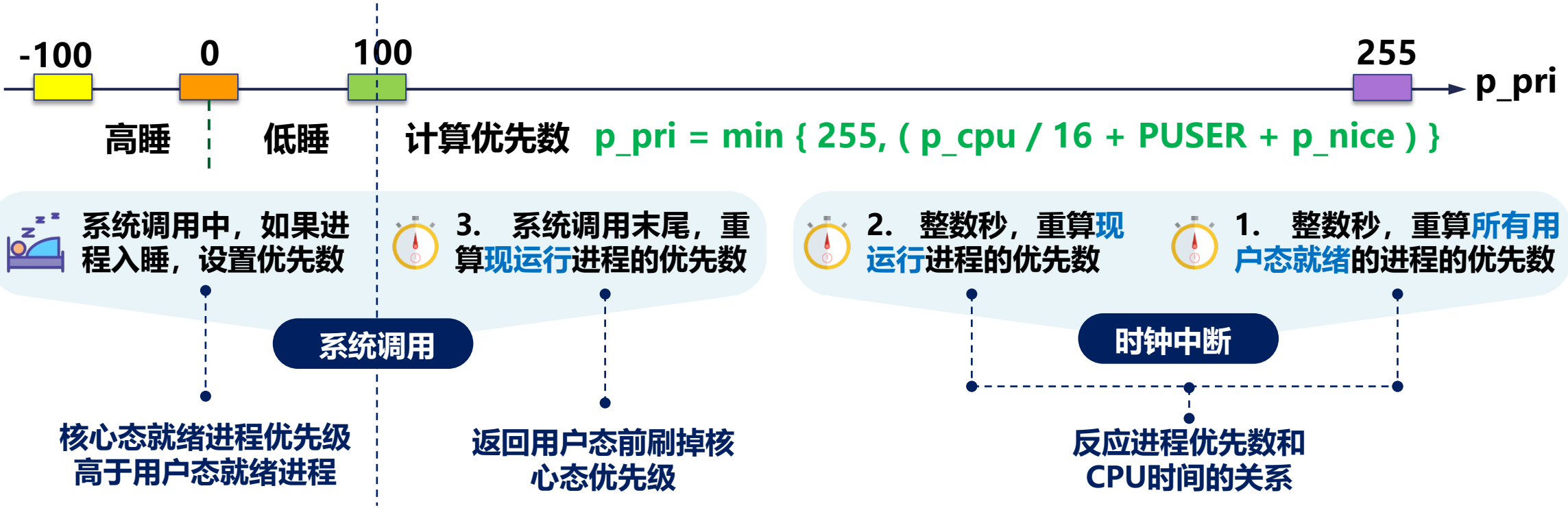
UNIX进程调度状态


一个用户态运行的进程PA
需要操作系统服务，如：
读写磁盘文件
进入核心态
INT 0x80
执行系统调用处理
如果系统调用中使用外设，进程调用Sleep入睡
下台1
Swtch进程切换
被唤醒
上台1
Swtch选中上台
（完成系统调用的剩余代码）
Trap末尾重算进程优先数
因为系统调用前为用户态
RunRun
下台2
Swtch进程切换
被Swtch选中上台
上台2
进程PB在执行
pb进入核心态
执行中断处理过程
返回用户态
设备中断

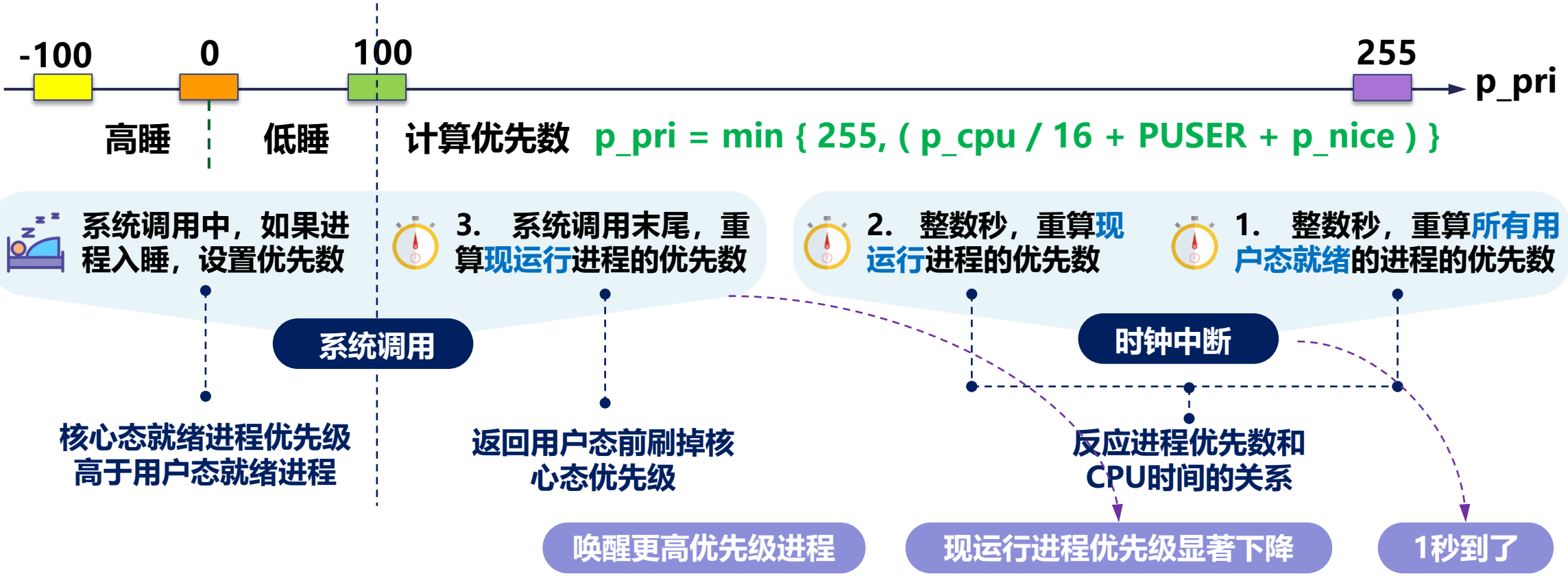
主要知识点

1. UNIX进程的Process类，User类和ProcessManager类
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

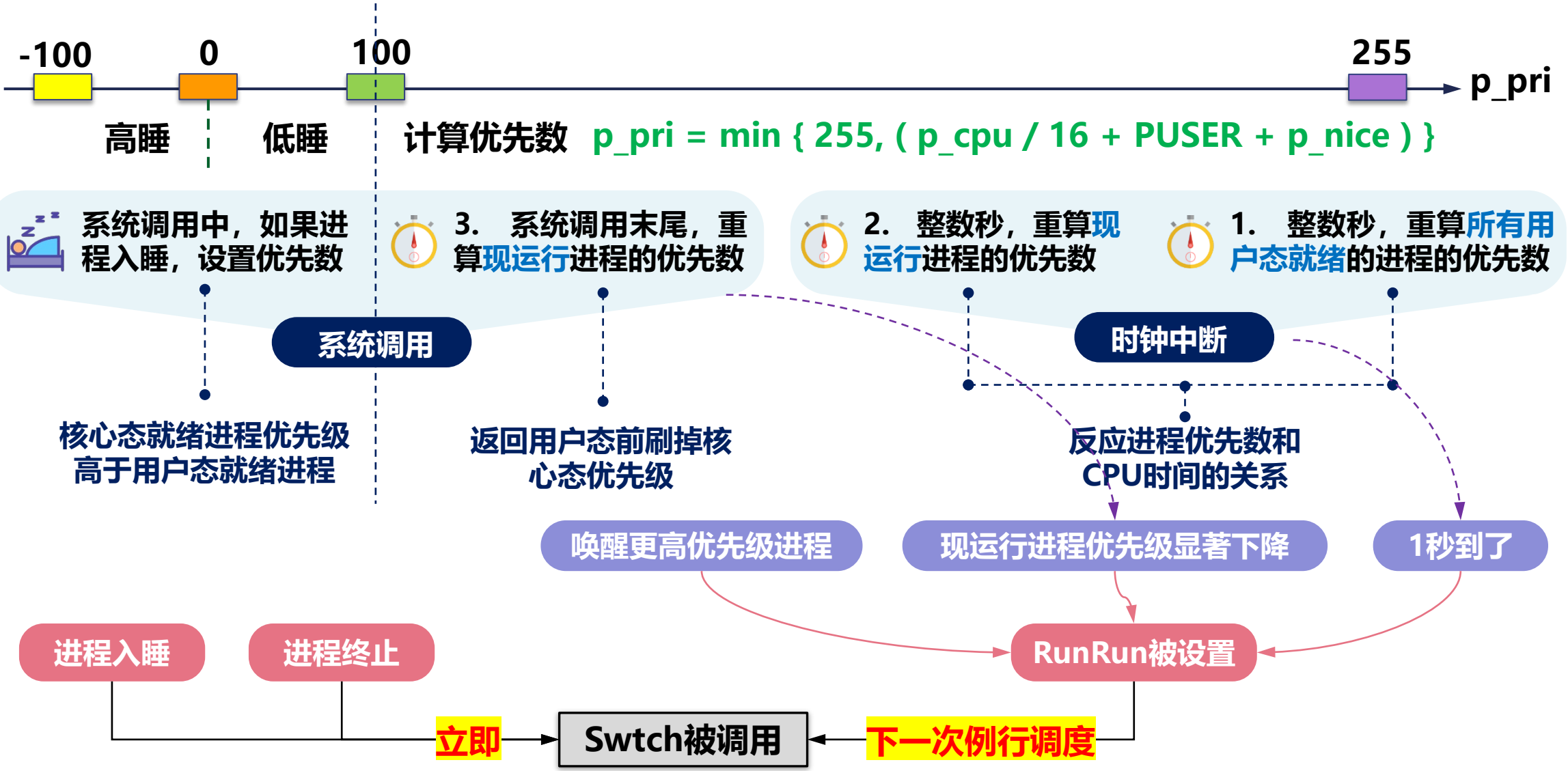
UNIX的动态优先权调度算法



UNIX的动态优先权调度算法



UNIX的动态优先权调度算法



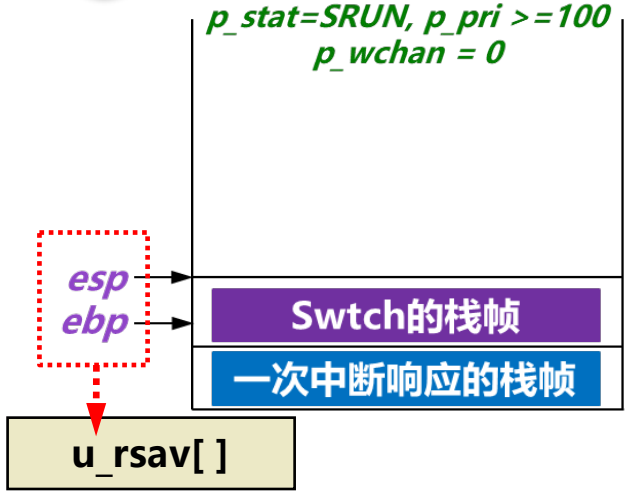
主要知识点

1. UNIX进程的Process类，User类和ProcessManager类
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

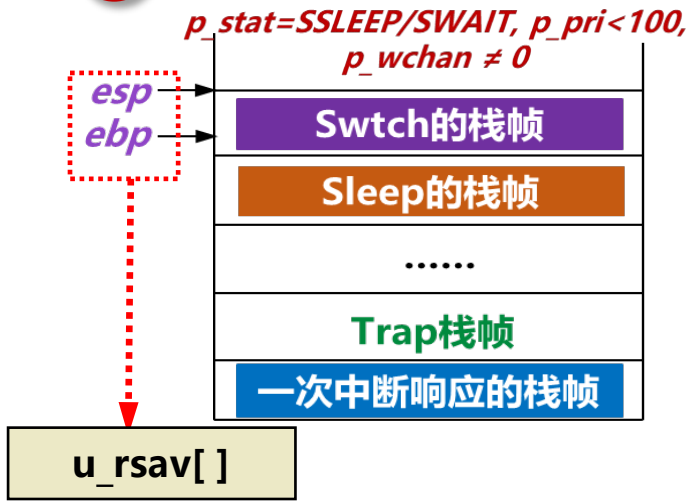
主要的内核函数

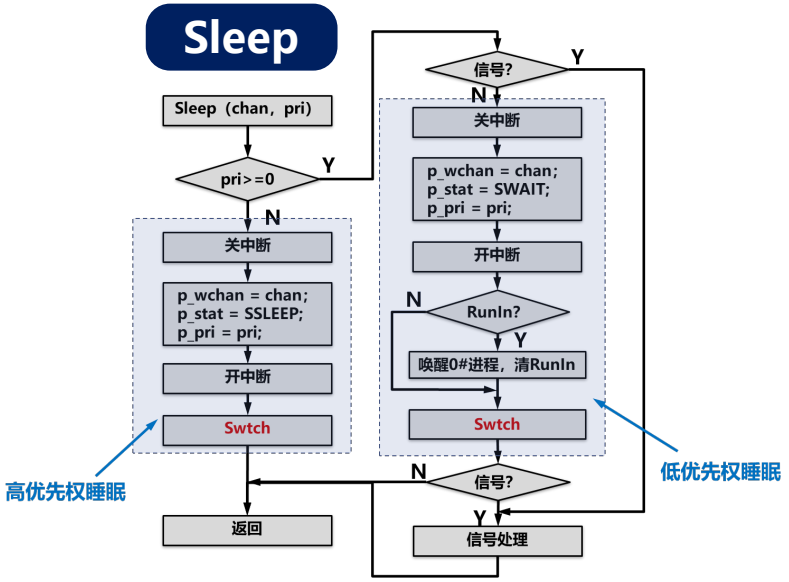
原进程保存现场后下台

1 被抢占进程的核心栈



2 非抢占进程的核心栈



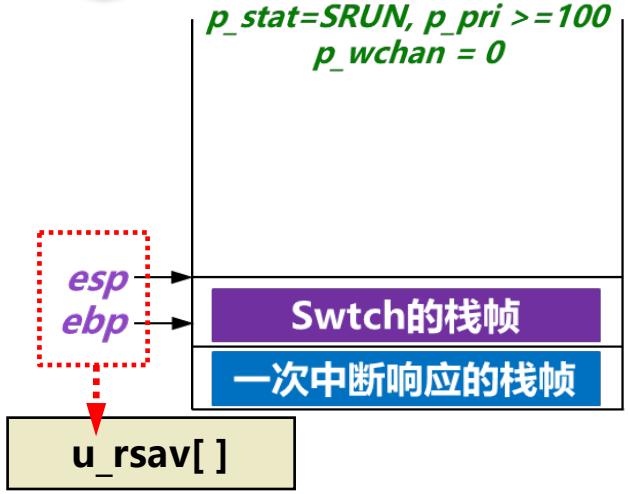


切换到0#的核心态页表，恢复0#现场

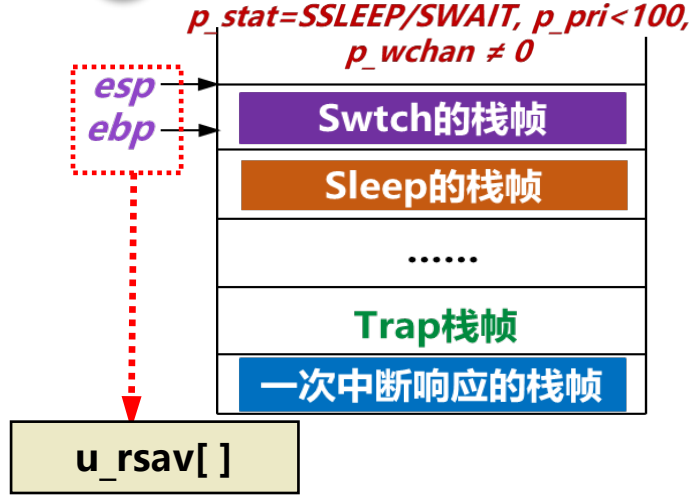
主要的内核函数

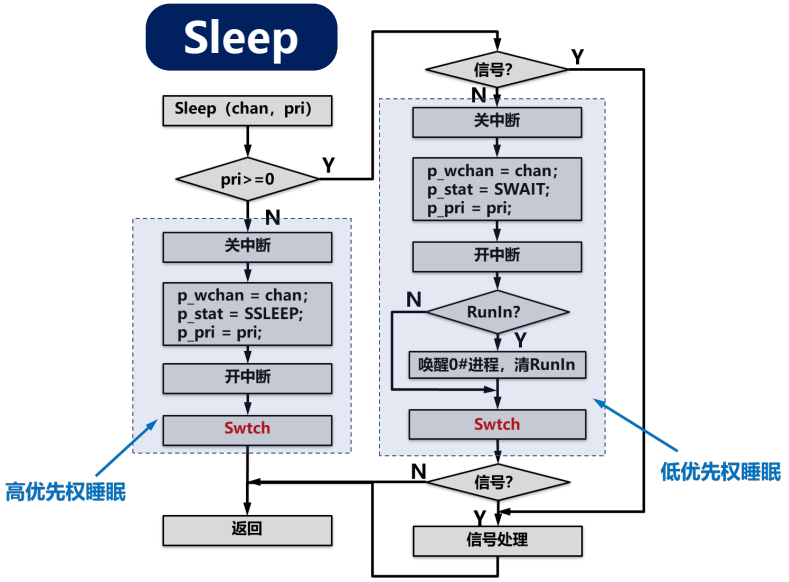
原进程保存现场后下台

1 被抢占进程的核心栈



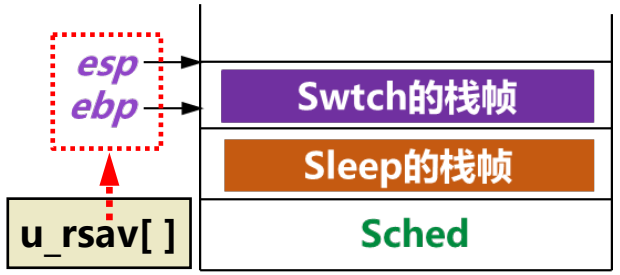
2 非抢占进程的核心栈





切换到0#的核心态页表，恢复0#现场

0#选新进程



0#：SSLEEP，-100

(Sched &RunIn或&RunOut)

- 有内存就绪进程：选优先级最高者（有可能仍是原进程）
- 无内存就绪进程：等中断

0#：SRUN，-100

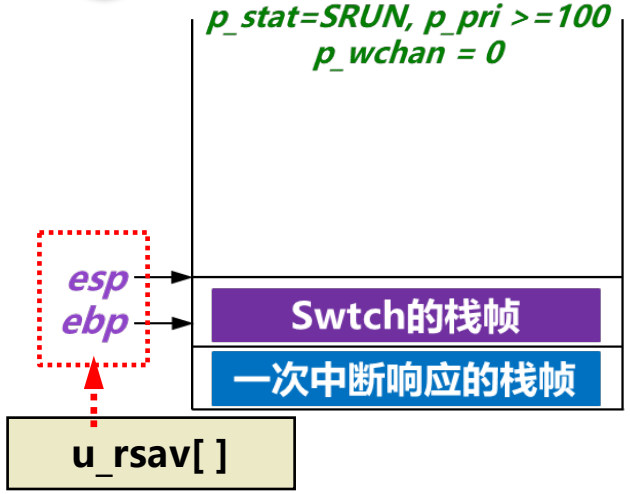
（RunOut时：唤醒一个盘交换区进程、
RunIn时：有进程低睡、1秒计时到）

就绪中的0#选到自己

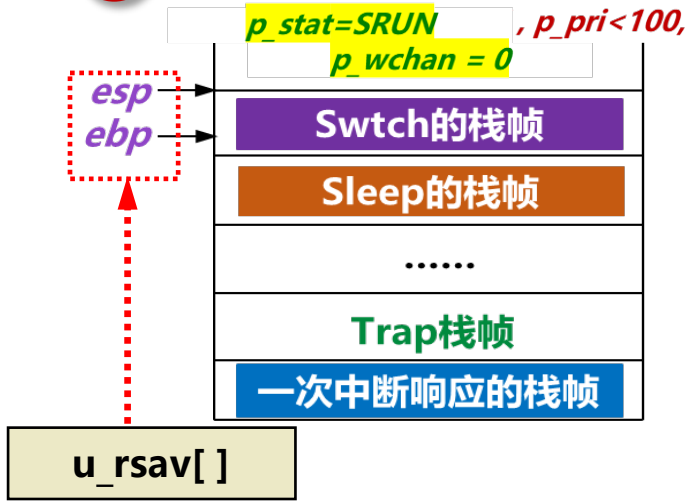
主要的内核函数

新进程恢复现场上台执行

1 被抢占进程的核心栈



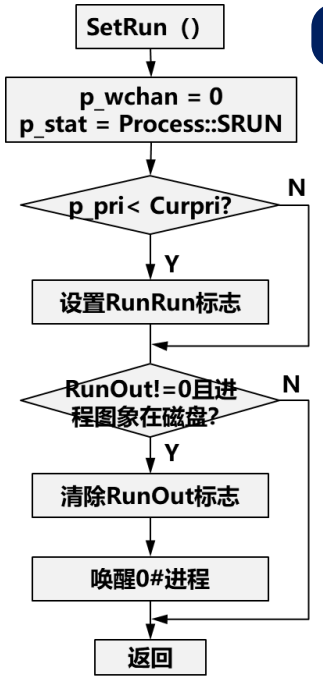
2 非抢占进程的核心栈



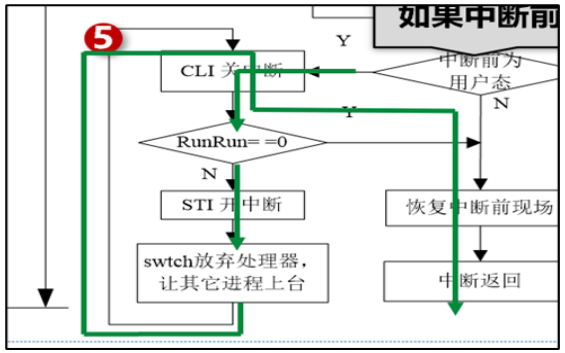
WakeUpAll

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

SetRun



执行Swtch中return 1



执行Swtch中的return 1

Sleep

.....

Trap

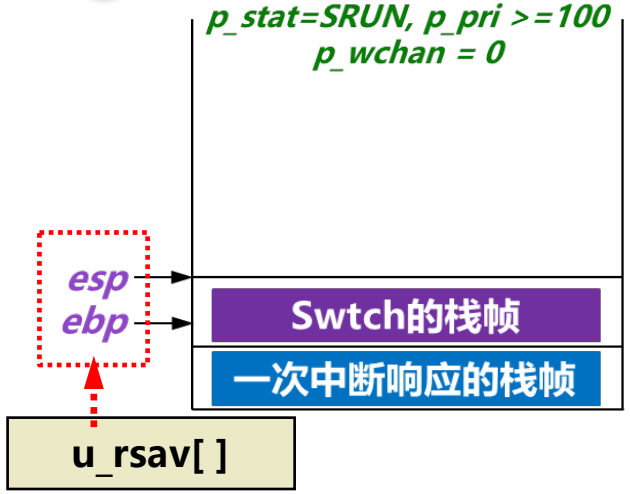
可能被抢占

返回用户态

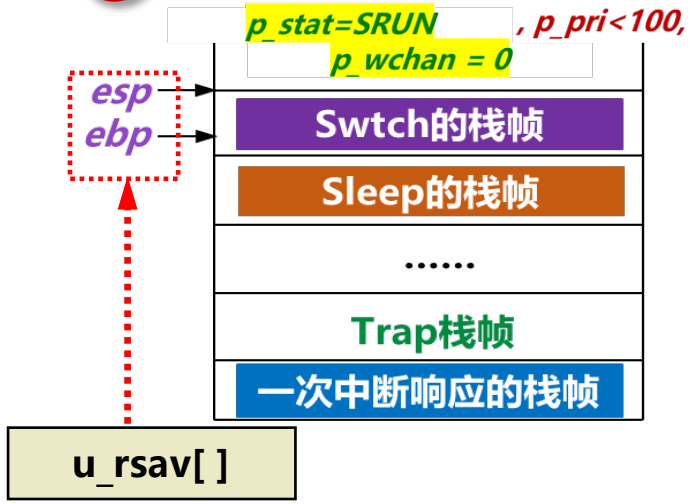
主要的内核函数

新进程恢复现场上台执行

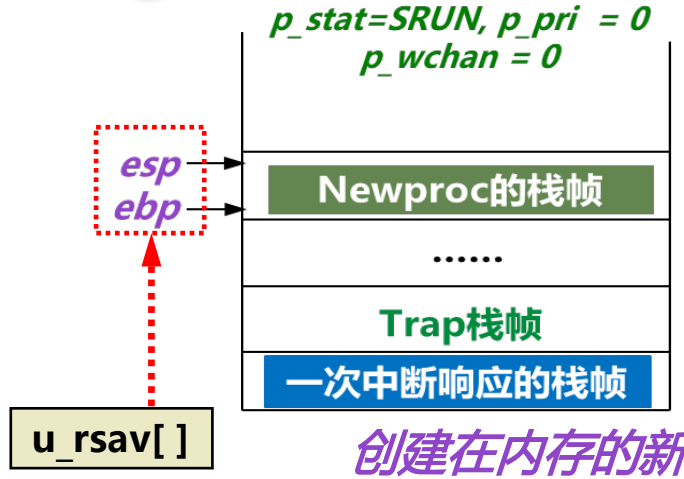
1 被抢占进程的核心栈



2 非抢占进程的核心栈

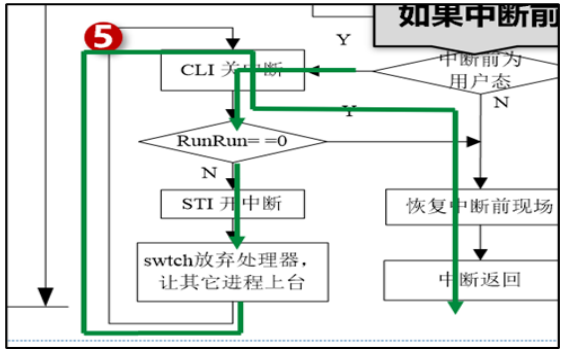


3.1 新进程的核心栈



创建在内存的新进程

执行Swtch中return 1



执行Swtch中的return 1

Sleep

.....

Trap

可能被抢占

返回用户态

执行Swtch中的return 1

NewProc

Fork

Fork

.....

Trap

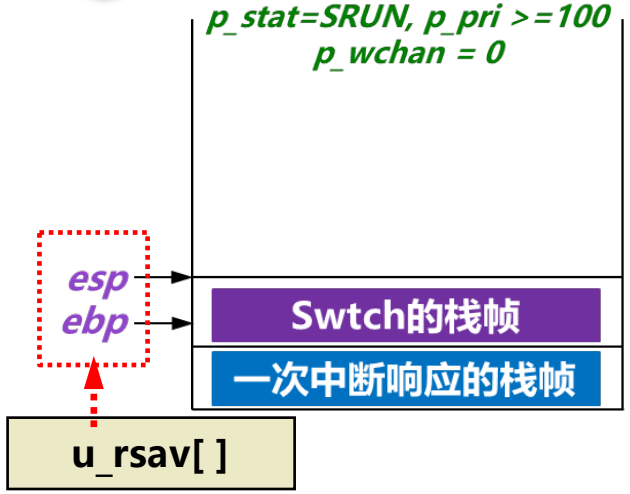
可能被抢占

返回用户态

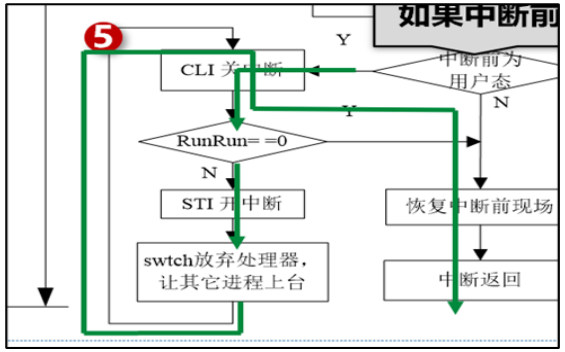
主要的内核函数

新进程恢复现场上台执行

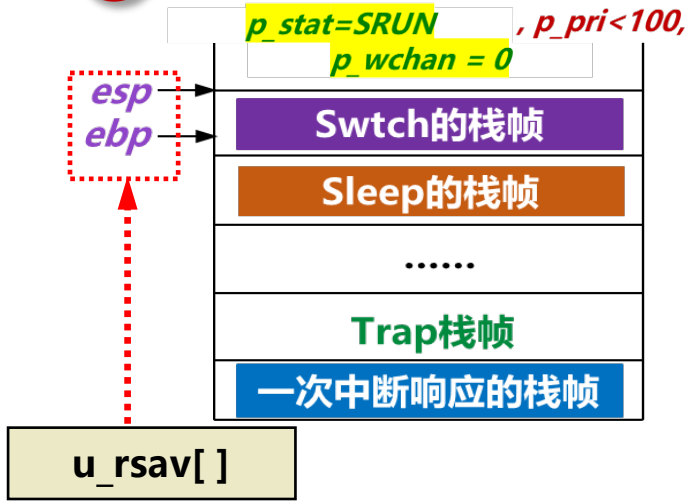
1 被抢占进程的核心栈



执行Swtch中return 1

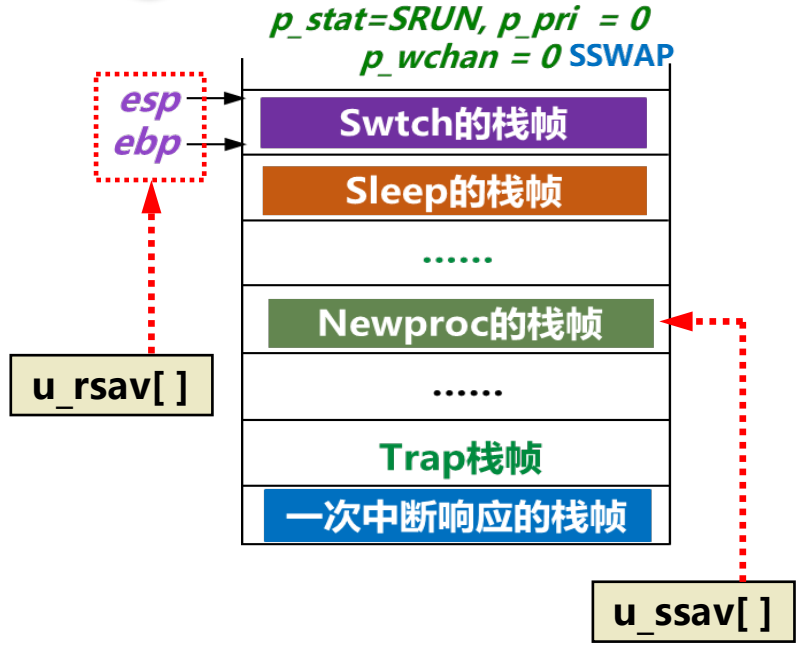


2 非抢占进程的核心栈



执行Swtch中的return 1

3.2 新进程的核心栈

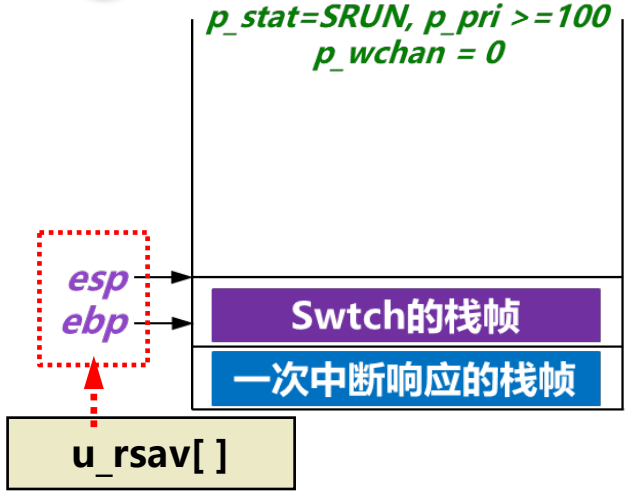


创建在盘交换区并已进入内存的子进程

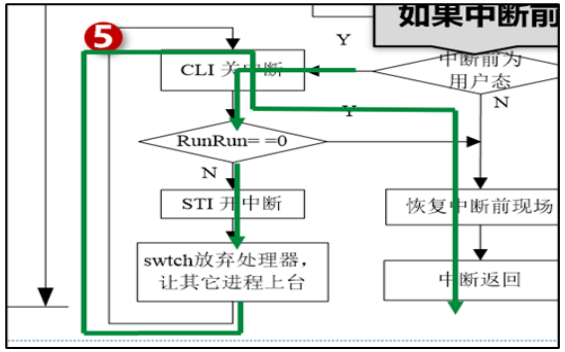
主要的内核函数

新进程恢复现场上台执行

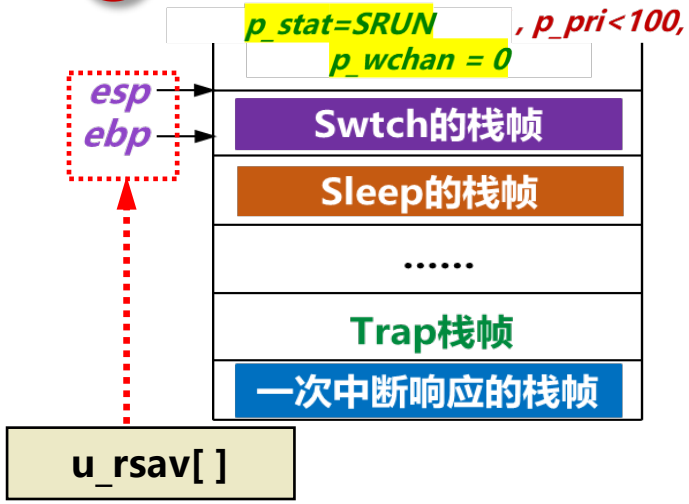
1 被抢占进程的核心栈



执行Swtch中return 1



2 非抢占进程的核心栈



执行Swtch中的return 1

Sleep

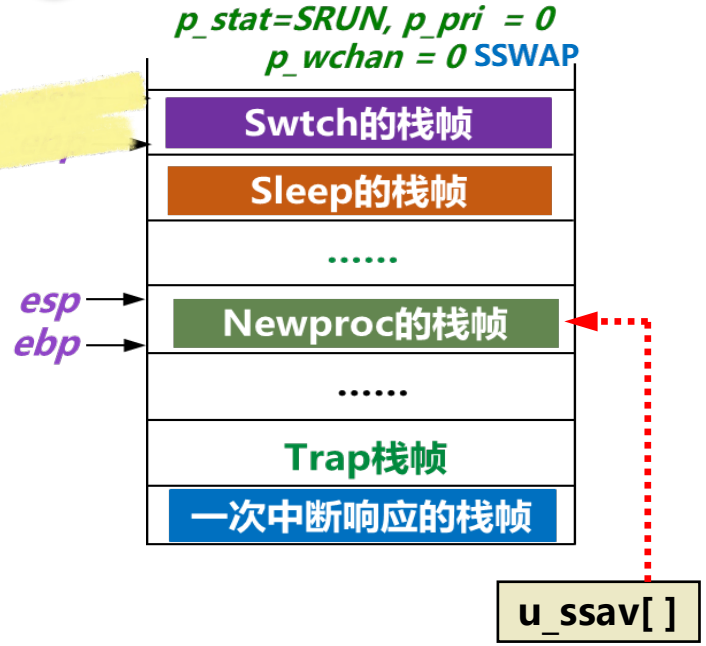
.....

Trap

可能被抢占

返回用户态

3.2 新进程的核心栈



创建在盘交换区并已进入内存的子进程

执行Swtch中的二次现场保护和return 1

和创建在内存的新进程返回路径相同

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 。

时钟中断与系统调用

1. 执行系统调用的过程涉及下列操作，其中由操作系统完成的是(**B**)。

Ⅰ. 保存断点和程序状态字

Ⅱ.保存通用寄存器的内容

Ⅲ.执行系统调用服务例程

Ⅳ.将 CPU 模式改为内核态

A. 仅Ⅰ、Ⅲ　　B. 仅Ⅱ、Ⅲ　　C. 仅Ⅱ、Ⅳ　　D. 仅Ⅱ、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 。

主要的内核函数

1. 下列关于UNIX V6++进程调度的描述中，正确的是： **C**

Ⅰ. 提供了抢占和非抢占两种调度方式

Ⅱ. 抢占调度发生在每一次中断即将返回用户态时

Ⅲ. 非抢占调度发生在进程入睡和终止时

Ⅳ. 非抢占调度和抢占调度下台的进程，核心栈的栈顶都是相同的

A. 仅Ⅰ　　B. 仅Ⅰ、Ⅱ　　C. 仅Ⅰ、Ⅲ、Ⅳ　　D. 全部正确

主要的内核函数

2. 假设某UNIX V6++系统中，在T0 时刻（T0 为整数秒）只有三个进程，分别是：现运行进程PA，内存中的高睡进程PB和内存中被抢占下台的就绪态进程PC，请回答：

（1）若T1时刻（T1=T0+1 秒，且T0 到T1 未发生进程切换），在用户态下执行的现运行进程PA响应时钟中断，待时钟中断结束后，PA、PB、PC 三个进程中，优先级增加的是__PC__，优先级减小的是__PA__，优先级不变的是__PB__ 。

（2）若T1 时刻（T1=T0+1 秒，且T0 到T1未发生进程切换），在用户态下执行的现运行进程PA响应设备中断唤醒进程PB，其间嵌套响应时钟中断，则时钟中断结束后，PA、PB、PC 三个进程中，优先级增加的是__无__，优先级减小的是__无__，优先级不变的是__PA，PB，PC__。

（填入你认为正确的进程名，如果没有，请填“无”，如有多个，请用“，”隔开）

主要的内核函数

3. 某UNIX V6++系统，时钟中断每秒钟发生60次，调度魔数SCHMAG是20。现运行进程和被剥夺进程优先数计算公式：p_pri = min {127 , 100 + （p_cpu/16）}。
T时刻整数秒，并发3个CPU密集型进程 PA、PB和PC，它们只计算，不执行系统调用，不启动IO操作。这3个进程静态优先数都等于100，PCB分别是Process[5]、[7]、[2]，PA先运行。

（1）请填写下表：

p_cpu/p_pri	T	T+1	T+2	T+3	T+4	T+5	……
PA	0/100	40/102	20/101	0/100	40/102	20/101	
PB	0/100	0/100	40/102	20/101	0/100	40/102	
PC	0/100	0/100	0/100	40/102	0/100	0/100	

主要的内核函数

（1）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：

主要的内核函数

- t0时刻现运行进程p2执行read系统调用：

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

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。

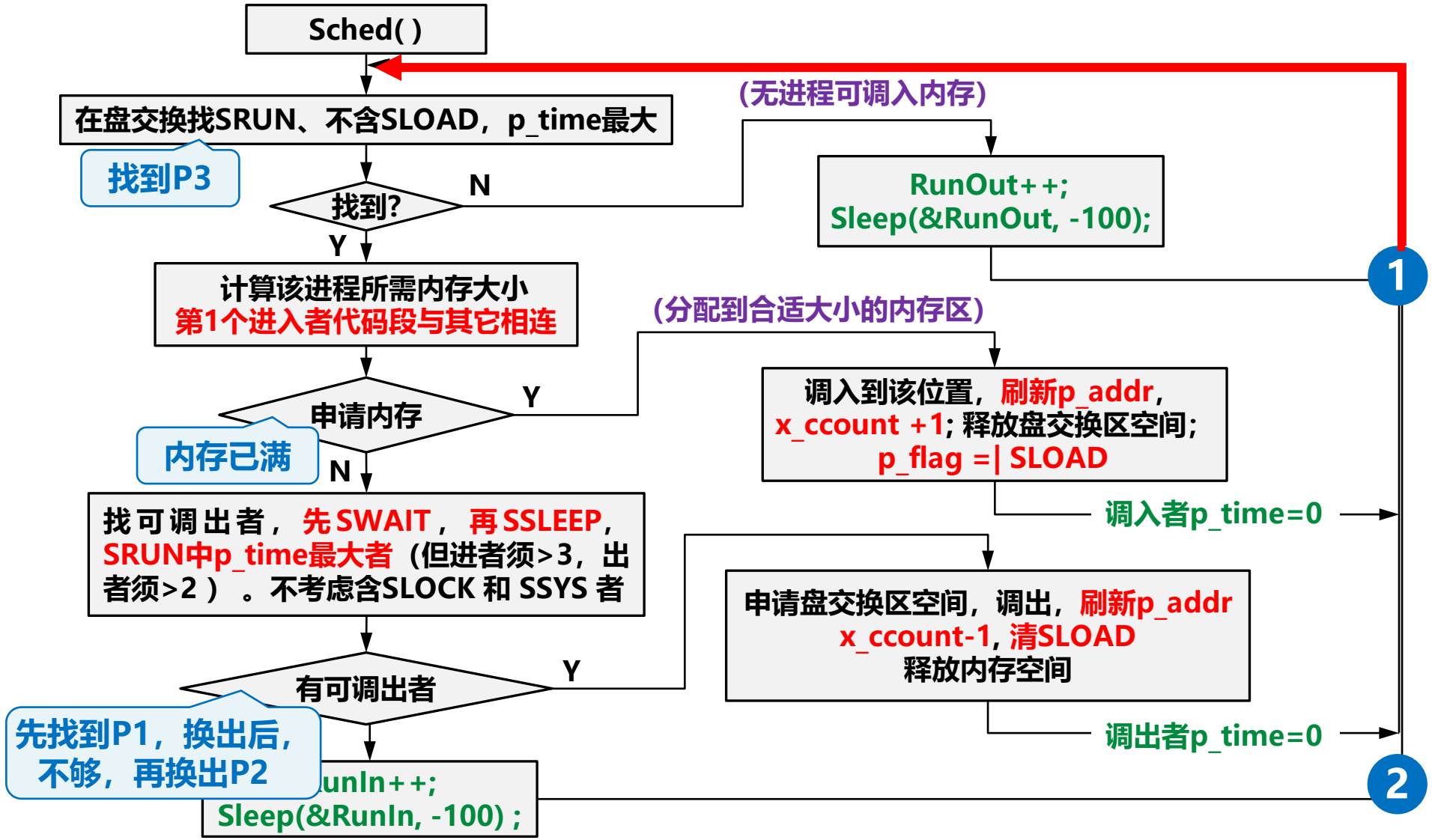
主要的内核函数

- 1秒后，p1在用户态执行，p3的I/O完：

序号	占用空间	优先数	状态	位置	年龄
0#	-	-100	执行	SLOAD	-
p1	40K	105	就绪	SLOAD	3
p2	60K	-50	高睡	SLOAD	3
p3	60K	10	就绪	~SLOAD	4

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。

主要的内核函数


Sched( )
在盘交换找SRUN、不含SLOAD，p_time最大
找到P3
找到?
N
Y
(无进程可调入内存)
RunOut++;
Sleep(&RunOut, -100);
1
计算该进程所需内存大小
第1个进入者代码段与其它相连
申请内存
Y
(分配到合适大小的内存区)
内存已满
N
调入到该位置，刷新p_addr，
x_ccount +1; 释放盘交换区空间;
p_flag =| SLOAD
调入者p_time=0
找可调出者，先SWAIT，再SSLEEP，
SRUN中p_time最大者（但进者须>3，出者须>2）。不考虑含SLOCK 和 SSYS 者
有可调出者
Y
申请盘交换区空间，调出，刷新p_addr
x_ccount-1, 清SLOAD
释放内存空间
调出者p_time=0
先找到P1，换出后，
不够，再换出P2
unIn++;
Sleep(&RunIn, -100) ;
2

主要的内核函数

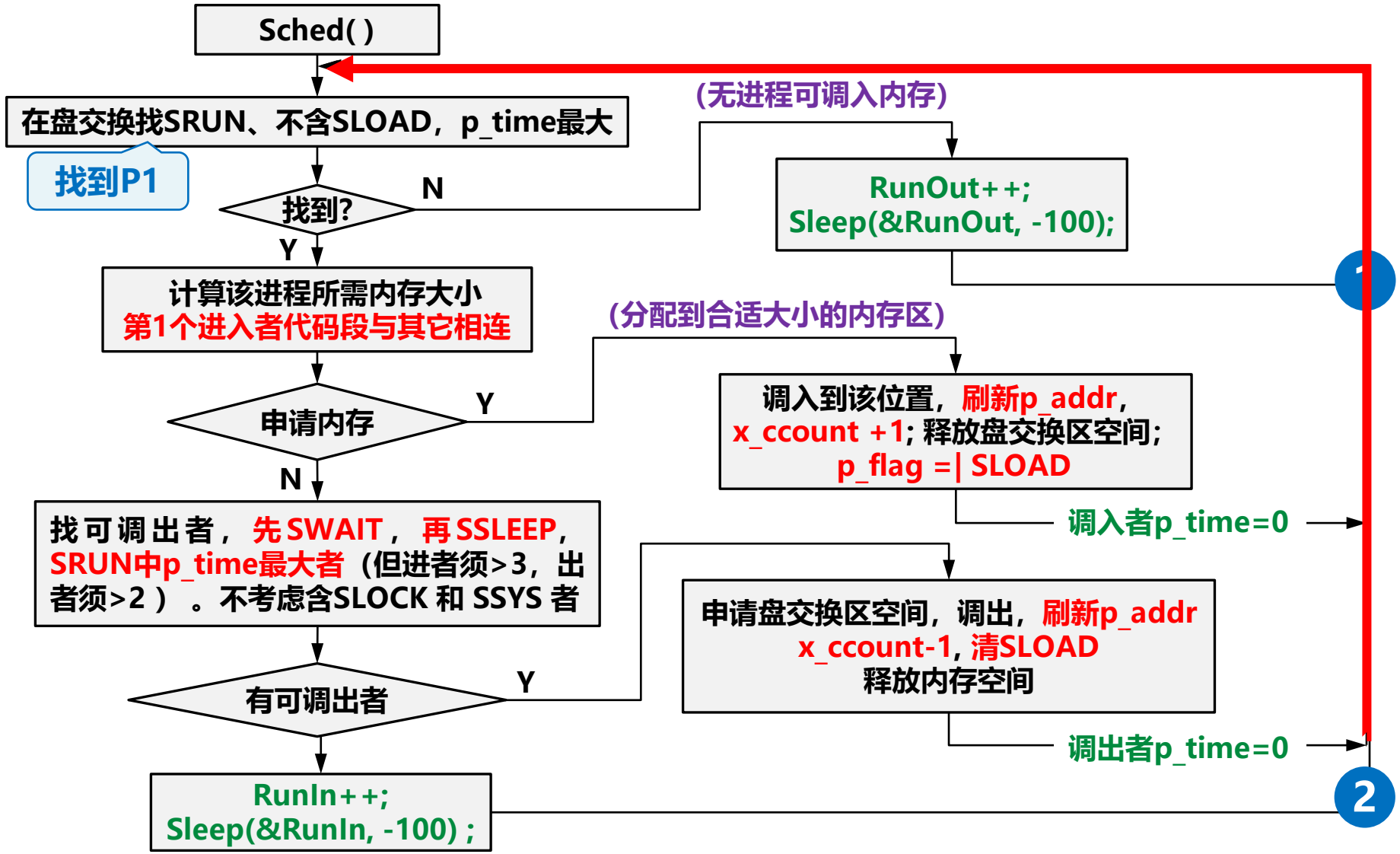
- 1秒后，p1在用户态执行，p3的I/O完：

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

0#选择p1换出，不够；

0#再选择p2换出，p3换入。

主要的内核函数


Sched( )
在盘交换找SRUN、不含SLOAD，p_time最大
找到P1
找到?
N
Y
(无进程可调入内存)
RunOut++;
Sleep(&RunOut, -100);
1
计算该进程所需内存大小
第1个进入者代码段与其它相连
申请内存
Y
(分配到合适大小的内存区)
调入到该位置，刷新p_addr，
x_ccount +1; 释放盘交换区空间;
p_flag =| SLOAD
调入者p_time=0
N
找可调出者，先SWAIT，再SSLEEP，
SRUN中p_time最大者（但进者须>3，出者须>2）。不考虑含SLOCK 和 SSYS 者
有可调出者
Y
申请盘交换区空间，调出，刷新p_addr
x_ccount-1, 清SLOAD
释放内存空间
调出者p_time=0
RunIn++;
Sleep(&RunIn, -100) ;
2

主要的内核函数

- 1秒后，p1在用户态执行，p3的I/O完：

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

0#找到p1，换入；无事可做，睡觉：P3上台。

主要的内核函数

（2）假设系统中只有上述代码运行，试回答下列问题：

```
#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);
   }
}
```

- 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.

- 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。

主要的内核函数

（2）假设系统中只有上述代码运行，试回答下列问题：

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

主要的内核函数

（2）假设系统中只有上述代码运行，试回答下列问题：

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

随堂练习

请写出下列代码的输出结果：

```
#include  <stdio.h>
int main (  )
{
    int i;
    printf ("%d  %d \n",  getpid ( ),  getppid ( ) );
    for (i = 0; i < 3; ++i)
        if ( fork( ) == 0 )
            printf ("%d  %d \n",  getpid ( ),  getppid ( ) );
    sleep(2) ;
    return 0;
}
```