




栈帧定位指针 ESP, EBP 存入自己的 user 结构, PA 成为普通的就绪态进程  
恢复现场时首先恢复核心态现场, PC 放弃 CPU 后, switch 函数选中 PA, 从它的  
user 结构中取出 T+1 秒保护的 switch 栈顶定位指针并赋值 ESP 和 EBP, 第二步  
恢复用户态现场, switch 函数返回, 栈帧出栈, PA 执行时钟中断处理入口函数  
例行调度之后的返回部分, 这一步会弹出时钟中断入口程序栈帧中保存的用户态  
寄存器, 之后 IRET, PA 返回用户态, 继续执行应用程序。

参考: PPT24 的表格和对这张 PPT 的讲解。



情景分析:  
假设系统中存在4个用户态的进程 PA、PB、PC、PD, 这些进程一直运算, 不IO, 不执行系统  
调用。进程的静态优先数相等: 100, p\_cpu是0。Process[5]、[7]、[8]、[9]分别是PA、PB、  
PC和PD进程的PCB。T时刻是整数秒, PA先运行。观察这些进程如何轮流使用CPU。

•  $p\_pri = \min \{127, \text{进程的静态优先数} + (p\_cpu/16)\}$


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

SCHMAG = 20  
HZ = 60

操作系统  
drong2004@tongji.edu.cn 15921642146

电信学院计算机系 邓蓉

24



## 习题的解答

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

SCHMAG = 20  
HZ = 60

- 时刻T+1, 整数秒时钟中断。被中断的现运行进程 PA 用户态运行。[T, T+1], PA连续使用CPU, 被时钟中断60次, p\_cpu=60。其余进程未运行, p\_cpu没有增加。T+1秒, 所有进程p\_cpu减20。PA、PB、PC、PD进程的p\_cpu值分别为40, 0, 0, 0。计算得进程优先级p\_pri分别为102, 100, 100和100。现运行进程PA的优先数增加了(原本是100, 现在是102), 用完时间片, runrun变1。
- 时钟中断返回用户态, 例行调度, runrun是1, PA进程让出CPU。系统选优先级最高的就绪态进程 PB, last\_select=PB。PB上台执行应用程序, 成为[T+1, T+2]时段的现运行进程。

在未来1秒之内, 一直是PB运行。每当系统发生时钟中断, PB就陷入核心态, 花一点点时间执行时钟中断处理程序。T+2秒, PB用尽时间片, 将CPU让出来。系统从last\_select+1开始遍历Process数组, 找优先权最高的就绪态进程。PC上台运行。

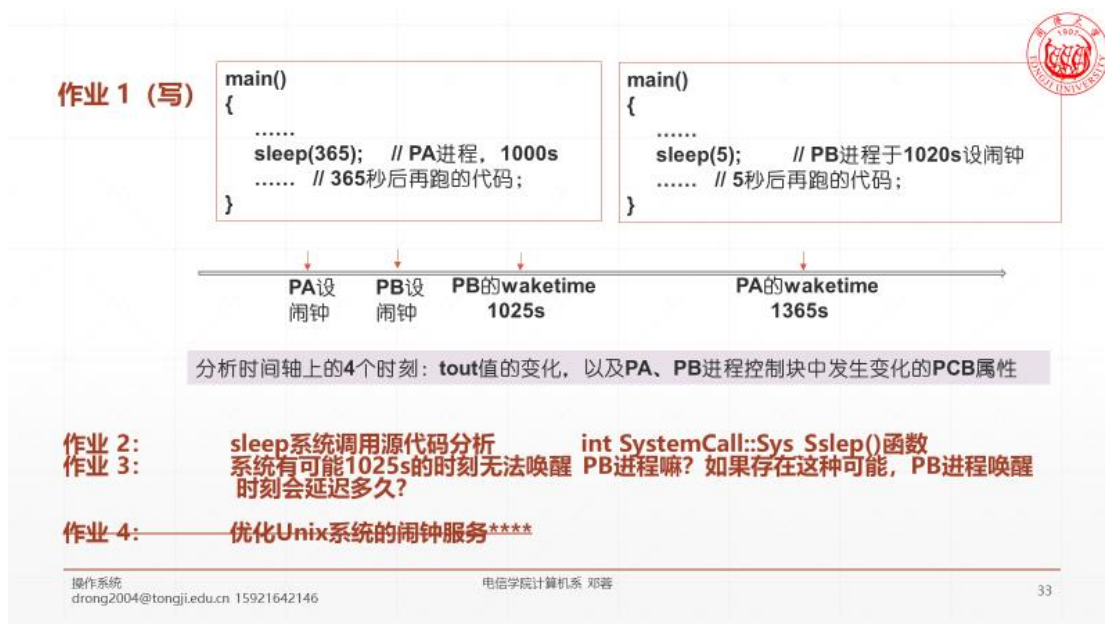
可以看到, 每4s是一个周期, PA、PB、PC、PD时间片轮转, 依次执行。第5s, PA进程已经连续3秒未得到运行, 优先数p\_pri已降至100, 成为优先级最高的进程, 再次得到运行。

操作系统  
drong2004@tongji.edu.cn 15921642146

电信学院计算机系 邓蓉

26

## Part 2、定时器服务



分析 1025 时刻 (1) 系统调度操作 (2) Sleep 系统调用下半部对 tout 变量的维护。

(1) 1025s, 整数秒进入时钟中断, 检查到 PB 进程设置的闹钟时间耗尽, 系统唤醒 PB 进程, PB 进程进入可执行队列, 等待系统选择它进入执行状态。

(2) Sleep 系统调用用于使进程进入休眠状态, 等待一定的时间。在 UNIX 系统中, 通常使用 tout 变量来维护所有进程的 waketime 的最小值。

tout 变量记录了下一个将唤醒的进程的触发时间。当一个进程调用 Sleep 时, 它会将 waketime 设置为当前时间加上休眠的时间。

如果一个进程设置了一个比当前 tout 更早的 waketime, 系统可能会更新 tout 的值, 以便在该时间点之前唤醒该进程。

(1) 假设 1025s 先运行进程时 PX, 运行在用户态

响应时钟中断, PX 执行时钟中断处理程序, 先前用户态, 调整系统时钟 time, 唤醒 waketime 到期的 PA, PB 进程。中断返回前, 例行调度, PX 优先级不及被唤醒的进程, 被剥夺, 放弃 CPU, 随后, PA、PB 分别执行自己的 sleep 系统调用后半段。

PA 执行系统调用后半段, waketime 未到期, 设置 tout (1365) 之后, 再次入睡

PB 执行系统调用后半段, waketime 到期, sleep 系统调用返回, 恢复应用程序优先数, 让出 CPU

这两个系统调用下半段全部执行完毕后, CPU 开始执行应用程序。PB, PX 还有其他进程时间片轮转, 直至有进程执行系统调用或 1365s 系统响应时钟中断唤醒 PA

系统调用下半段执行期间, 有可能 PA 先运行, 也有可能 PB 先运行 (1) 如果 PA 先运行, 入睡后 PB 执行 sleep 系统调用下半段, 完成后, 系统开始执行应用程序 (2) 如果 PB 先运行, 返回用户态前让出 CPU 给 PA 执行系统调用下半段, PA 入睡后系统开始执行应用程序,

PB、PX 和其它进程时间片轮转。

(2) 假设 1025s 先运行进程 PY 运行在核心态

PY 响应时钟中断，先前是核心态，直接返回，不调整系统时钟 time，不会唤醒 PA，上述操作延迟直至先前是用户态的第一个时钟中断

(3) 1025s，系统 idle

执行上述红色操作。时钟中断处理程序执行完毕后，0#进程选中 PA 或 PB，让出 CPU，恢复睡眠等待 RunIn 或 RunOut 的状态。

## Part 3、综合题

全嵌套中断处理模式。低优先级中断处理程序运行时，系统响应高优先级中断处理请求。已知，时钟中断优先级高于磁盘中断优先级。假设：900s，PA 进程执行 sleep (100) 入睡。998s，PB 进程执行 read 系统调用，读磁盘文件。1000s，现运行进程 PX 正在执行应用程序。PA 设置的闹钟到期、PB 读取的磁盘文件数据 IO 结束。分析 1000s，系统详细的调度过程。分两种情况考虑：

### 1、先响应磁盘中断

T=1000s： 现运行进程为 PX。此时，PA 的 sleep(100)中断尚未到期，PB 正在执行 read 系统调用，导致磁盘 I/O 操作，而时钟中断优先级高于磁盘中断。

T=1001s： 由于磁盘中断请求，系统响应了磁盘中断。中断处理程序可能会暂停 PB 进程的执行，保存 PB 的执行现场，并开始处理磁盘中断。在中断处理期间，时钟中断可能被屏蔽，以防止中断嵌套。

T=1002s： 磁盘中断处理完毕后，系统可能会进行调度决策。由于此时 PA 的 sleep(100)中断到期，PA 可能被唤醒，被移动到可运行队列。由于时钟中断被屏蔽，时钟中断不会打断这一过程。

T=1003s： 调度程序选择运行 PA，PA 开始执行，PB 仍然处于阻塞状态。

总体而言，由于磁盘中断的优先级较低，系统在响应磁盘中断后，会立即检查更高优先级的时钟中断，确保及时唤醒 PA 进程。然后，PA 进程得到运行机会。

### 1、先响应磁盘中断

1000s。内核任务执行次序：磁盘中断处理程序。。被打断，时钟中断处理程序（没有调整 time，不会唤醒 PA），磁盘中断处理程序恢复执行，read 系统调用后半段。全部完成后，PX、PB 轮流使用 CPU 执行应用程序。

1000s 之后，先前是用户态的第一次时钟中断。内核任务执行次序：时钟中断处理程序（调整 time，唤醒 PA），sleep 系统调用后半段。全部完成后，PX、PA、PB 轮流使用 CPU 执行 应用程序。

具体调度细节：

1000s。 现运行进程 PX 执行磁盘中断处理程序，唤醒 PB、RunRun++。其间，嵌套执行中断优先权更高的时钟中断处理程序。后者先前核心态，不调整 time，不会唤醒 waketime 到期的 PA。磁盘中断处理程序运行结束后，现运行进程 PX 被剥夺、放弃 CPU。SRUN 进程集合 {PX, 优先数>=100; PB, 优先数=-50}。PB 优先级高，被选中，执行 read 系统调用后半段。完成后恢复其应用程序优先数 (p\_pri >= 100)，让出 CPU。SRUN 进程集合 {PX, 优先数>=100; PB, 优先

数 $\geq 100$  }。PB 和 PX 轮流使用 CPU 执行应用程序。

1000s 之后，先前是用户态的第一次时钟中断。现运行进程执行时钟中断处理程序，调整 time，唤醒 PA、RunRun++。时钟中断处理程序运行结束后，现运行进程被剥夺、放弃 CPU。SRUN 进程集合 {PX, 优先数 $\geq 100$ ; PA, 优先数 $=90$ ; PB, 优先数 $\geq 100$  }。PA 优先级最高，被选中，执行 sleep 系统调用后半段。waketime 到期，PA sleep 系统调用返回，恢复应用程序优先数 ( $p\_pri \geq 100$ ) 之后，让出 CPU。SRUN 进程集合 {PX, 优先数 $\geq 100$ ; PA, 优先数 $\geq 100$ ; PB, 优先数 $\geq 100$  }。PA、PB、PX 轮流使用 CPU 执行应用程序

## 2、先响应时钟中断

T=1000s: 现运行进程是 PX。此时，PA 的 sleep(100)中断已经到期，可能触发时钟中断。

T=1001s: 由于 PA 的 sleep(100)中断到期，系统可能响应时钟中断。中断处理程序会保存 PX 的执行现场，并进行时钟中断的处理。在中断处理过程中，PB 的 read 系统调用可能仍在进行。

T=1002s: 时钟中断处理完毕后，系统可能会进行调度决策。由于 PA 的 sleep(100)中断已经到期，PA 可能会被唤醒，移动到可运行队列。此时，系统可能选择运行 PA，因为时钟中断已经得到响应，可能继续 PX 的执行。PB 的 read 系统调用可能仍在进行。

T=1003s: PA 进程得到运行机会，继续执行。PB 的 read 系统调用可能仍在进行，PX 也可能继续执行。

总体而言，在这种情况下，由于 PA 的 sleep(100)中断到期，系统先响应时钟中断，保护 PX 的执行现场。然后，系统可能选择运行 PA 进程，继续 PX 的执行。PB 的 read 系统调用可能仍在进行。这种情况下，时钟中断的处理被插入到了两个不同进程的执行中。

1000s，内核任务执行次序：时钟中断处理程序。。。被打断，硬盘中断处理程序，时钟中断处理程序恢复执行，read 系统调用后半段，sleep 系统调用后半段。全部完成后，PX、PA 和 PB 轮流使用 CPU 执行应用程序。

具体调度细节：现运行进程 PX 执行时钟中断处理程序，更新 Time::time，STI、EOI 后，中断嵌套，响应 磁盘中断。磁盘中断处理程序唤醒 PB、RunRun++。先前核心态，不调度，磁盘中断处理程序运行结束后，PX 返回、继续执行被打断的时钟中断处理程序。时钟中断处理程序唤醒 PA、RunRun++(值是 2)。先前用户态，例行调度。RunRun 非 0，现运行进程 PX 被剥夺、放弃 CPU。SRUN 进程集合 {PX, 优先数 $\geq 100$ ; PA, 优先数 $=90$ ; PB, 优先数 $=50$  }。PB 优先级最高，被选中，执行 read 系统调用后半段。完成后恢复其应用程序优先数 ( $p\_pri \geq 100$ )，让出 CPU。SRUN 进程集合 {PX, 优先数 $\geq 100$ ; PA, 优先数 $=90$ ; PB, 优先数 $\geq 100$  }。PA 优先级最高，被选中，执行 sleep 系统调用后半段。waketime 到期，PA sleep 系统调用返回，恢复应用程序优先数之后，让出 CPU。SRUN 进程集合 {PX, 优先数 $\geq 100$ ; PA, 优先数 $\geq 100$ ; PB, 优先数 $\geq 100$  }。PA、PB、PX 轮流使用 CPU 执行应用程序