时钟中断和时间片轮转调度

同济大学计算机系　操作系统作业　　2023-11-20
学号　2154312　　姓名　郑博远

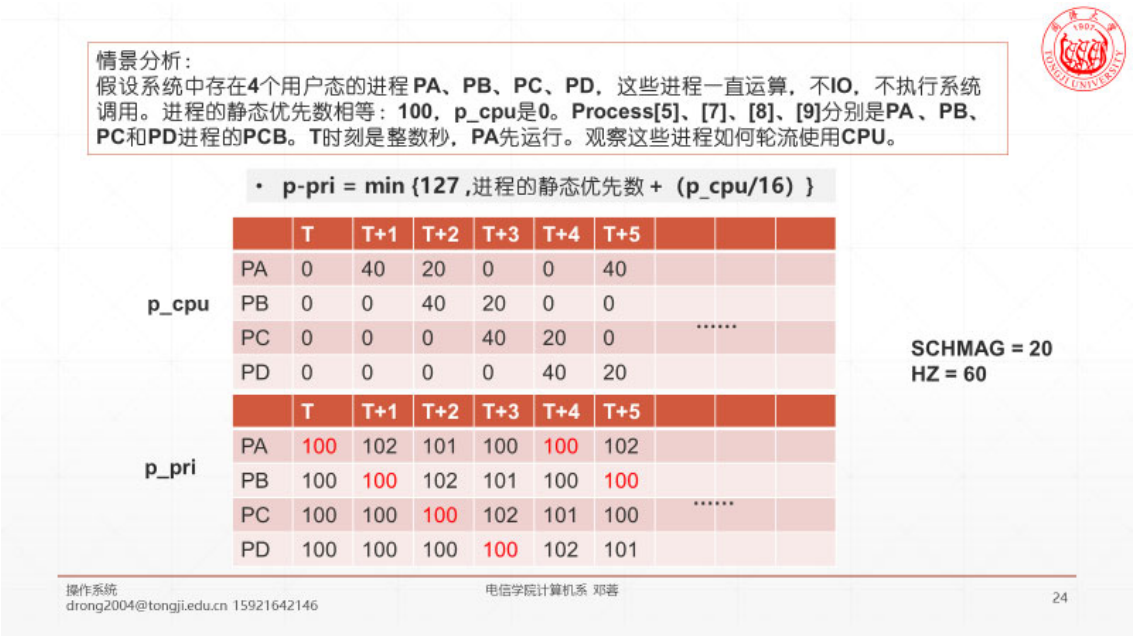
Part 1、Unix V6 时间片轮转调度的实现

习题： Unix V6++系统中存在 3 个 CPU bound 用户态进程 PA、PB 和 PC。3 个进程静态优先数相等：100，p_cpu 是 0。Process[8]、[5]、[9]分别是 PA 、PB、PC 进程的 PCB。T 时刻是整数秒，PA 先运行。

1、 画进程运行时序图。

2、 T+1 时刻，PA 进程用完时间片放弃 CPU。何时，PA 进程会再次得到运行机会？简述 T+1 时刻系统怎样保护 PA 进程的用户态 CPU 执行现场，下次再运行时系统如何恢复 PA 进程的用户态 CPU 执行现场。

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 ,进程的静态优先数 + (p_cpu/16) }

p_cpu

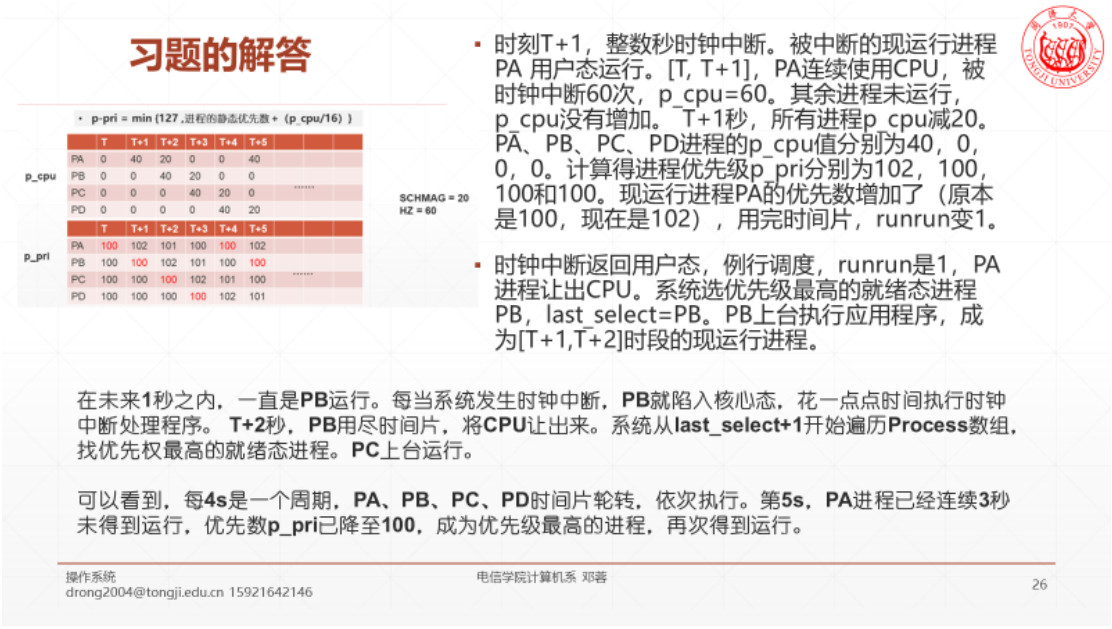
	T	T+1	T+2	T+3	T+4	T+5			
PA	0	40	20	0	0	40			
PB	0	0	40	20	0	0			
PC	0	0	0	40	20	0	.....		
PD	0	0	0	0	40	20			

SCHMAG = 20
HZ = 60

p_pri

	T	T+1	T+2	T+3	T+4	T+5			
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			

操作系统
drong2004@tongji.edu.cn 15921642146
电信学院计算机系 邓蓉
24



答：（1） （本小题答案正确）

p_cpu:

	T	T+1	T+2	T+3	T+4	T+5	
PA	0	40	20	0	40	20	……
PB	0	0	0	40	20	0	……
PC	0	0	40	20	0	40	……

p_pri:

	T	T+1	T+2	T+3	T+4	T+5	
PA	100	102	101	100	102	101	……
PB	100	100	100	102	101	100	……
PC	100	100	102	101	100	102	……

（2）T+3 时刻，PA 进程再次运行。当发生时钟中断时，整数秒会重新计算每个进程的 p_pri。若此时 PA 进程的优先级最高（即对应 T+3 等时刻，3s 一个轮回），高于当前执行的进程，则将 RUNRUN++。此时在返回中断执行例行调度时，就会选择 p_pri 最高的进程 PA 上台。

T+1 时刻保护现场，T+3 时刻恢复现场，具体操作如下：

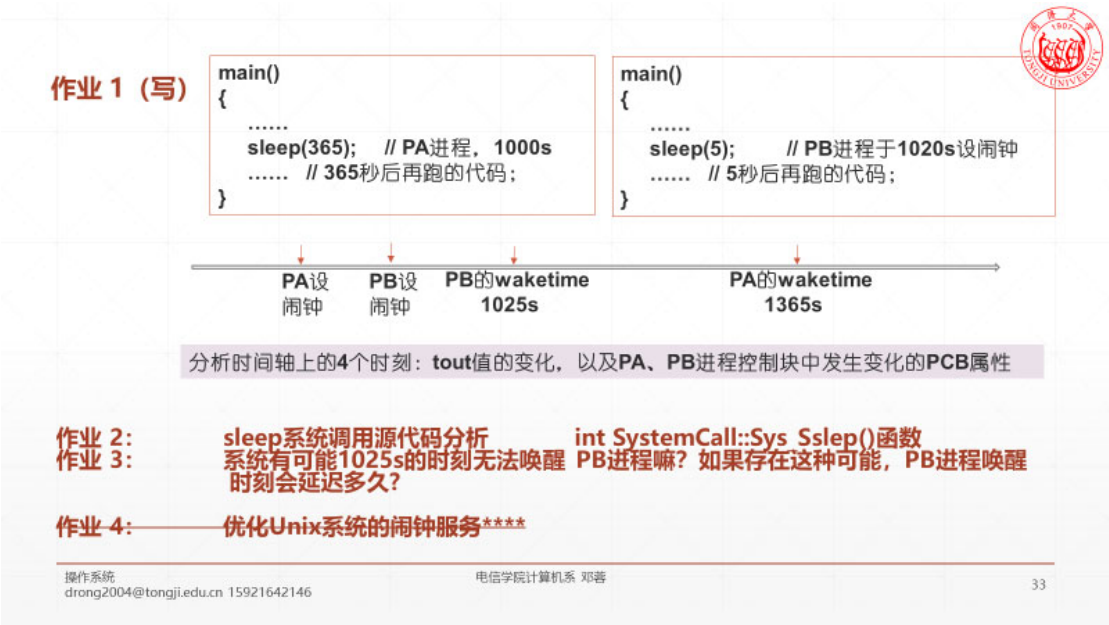
1.第一次现场保护：中断响应过程中将 PA 的用户态寄存器压入到自己的核心栈；

2.第二次现场保护：当 PA 进程放弃 CPU 下台时，PA 执行 swtch()放弃 CPU，系统通过宏 SaveU 将其 ESP 与 EBP 寄存器保存到 User 结构中的 u_rsav 数组中来保护现场。

3.恢复现场第 1 步:当 PA 进程上台恢复现场时,系统通过宏 RetU 将 User 结构中 u_rsav 数组中取出之前的 esp 与 ebp 值，赋值 ESP、EBP 寄存器。

4.恢复现场第 2 步：swtch 函数返回，swtch 栈帧出栈。PA 执行时钟中断入口函数例行调度之后的中断返回部分。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 回到用户态继续执行应用程序。

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。

答：

作业 1：

时刻 1（1000s），PA 进程将 tout 设为 1365，wakeTime = 1365，PA 进程的 p_stat=SWAIT，p_pri=90，p_wchan=&Time::tout;

时刻 2（1020s），PB 进程将 tout 设为 1025 ，wakeTime = 1025，PB 进程的 p_stat=SWAIT，p_pri=90，p_wchan=&Time::tout;

时刻 3（1025s），PA、PB 被唤醒。此时 PA、PB 的 PCB：p_stat = SRUN，p_wchan=0，p_pri 不变仍为 90

PB 进程调度上台执行 sleep 系统调用的后半部分：wakeTime 到期，系统调用完成。返回用户态前 PB 执行 setPri()修正的~~p_stat=SRUN，~~p_pri>=100~~，p_wchan=0，~~。

PA 上台后将 tout 设为 1365，继续入睡（p_pri=90，p_stat=SWAIT，p_wchan=&Time::tout）；

时刻 4（1365s），PA 进程的 p_stat=SRUN，p_pri>=100，p_wchan=0，直接从系统调用返回，tout 不变。

作业 2：

```
int SystemCall::Sys_Sslep()
{
    // 获取当前进程的 User 结构
    User& u = Kernel::Instance().GetUser();

    // 关中断
    X86Assembly::CLI();

    // 计算进程的唤醒时间
    unsigned int wakeTime = Time::time + u.u_arg[0];    /*
sleep(second) */

    // 若进程还未到唤醒时间就被叫醒，则继续入睡；否则直接开中断出系统调用
    while( wakeTime > Time::time )
    {
        if ( Time::tout <= Time::time || Time::tout > wakeTime )
        {
            // 更新维持 tout 是最小的 wakeTime
            Time::tout = wakeTime;
        }
        // 没到时间，继续睡
        u.u_procp->Sleep((unsigned long)&Time::tout,
ProcessManager::PSLEP);
    }

    // 开中断
    X86Assembly::STI();

    return 0;   /* GCC likes it ! */
}
```

作业 3：① 1025s，CPU 关中断；② 若 1025s 的时钟中断响应时正处在核心态，则进入时钟中断处理程序后，由于先前是核心态，不进入后续的繁琐事务处理，即不唤醒 PB。因此，PB 进程的唤醒需要延迟到下一个最近的响应中断前为用户态进程的时钟滴答~~整数秒~~。

作业 4：考虑 ProcessManager 维护一个小根堆，堆中的每个元素都是一个结构体，包含 wakeTime 和指向 PCB 的指针这两个元素，以 wakeTime 作为排序依据。每次整数秒进入时钟中断时，比较当前的 Time::time 与堆顶元素的 wakeTime，若等于则持续弹出直至 Time::time<堆顶 wakeTime，并直接用 PCB 指针 setRun 对应的进程。就无需唤醒所有的睡眠进程，也无需扫描整个 PCB 表来找到对应 wchan 的进程。

分析：

(1) 1025s 时刻，时钟发出中断。在时钟中断处理程序中发现 Time::time==Time::tout，唤醒所有延时睡眠的进程。中断处理程序返回后，执行例行调度选择新进程上台。执行 PA 进程的系统调用后半段时，由于 wakeTime>Time::time，继续入睡放弃 CPU。执行 PB 进程的系统调用后半段时，wakeTime==Time::time 不入睡。系统调用返回时例行调度，PB 进程在用户态上台，执行 PB 进程 sleep()5s 后的代码。

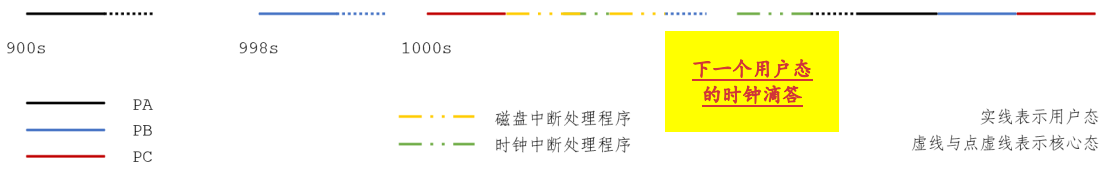
(2) Sys_Sslep 系统调用下半段，PB 进程不满足 wakeTime > Time::time，因此不维护 Time::tout，系统调用结束；PA 进程满足 wakeTime<Time::time，此时 Time::tout <= Time::time，将 Time::tout 更新为 PA 进程的 wakeTime，即 1365。

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
2、 先响应时钟中断

答：1.



1000s 时，CPU 响应中断，现运行进程 PX 用户态运行。中断优先级是 0，响应磁盘中断请求；随后，时钟中断请求中断优先级高于磁盘中断请求，中断嵌套，磁盘中断处理程序被打断，系统执行时钟中断处理程序。由于先前为核心态，时钟中断处理程序只负责 Time::lbolt 与 u_stime 的维护，不执行后半段繁琐的事务处理，因此不唤醒进程 PA。时钟中断处理程序执行完毕后，先前核心态不调度，不考虑剥夺 PX 进程，它继续运行磁盘中断处理程序后半段，唤醒进程 PB。执行完毕后，由于先前为用户态，执行例行调度。

候选进程中，{PX， p_pri>=100； PB， p_pri = -50}。选择 PB 上台，Swtch→Sleep→Sys_read 执行 read 系统调用后半段。随后 Sys_read→Trap，PB 优先级由-50 升至大于等于 100,RunRun++让出 CPU。Trap→SystemCallEntrance,RunRun 非 0，执行例行调度。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~~在下一次整数秒到来时~~，CPU 当前进程为用户态，响应时钟中断进入时钟中断处理程序，因此会执行后半段，因此唤醒进程 PA，RunRun++。执行完毕后，由于先前为用户态，执行例行调度。候选进程中，{ PX，p_pri>=100； PB，p_pri >= 100； PA，p_pri = 90 }。选择 PA 上台，Swtch→Sleep→Sys_Sslep 执行时钟系统调用后半段。随后 Sys_read→Sslep，PA 优先级由 90 升至大于等于 100。Trap→SystemCallEntrance，RunRun 非 0，执行例行调度。

最后，3 个用户态进程，PA、PB、PX 时间片轮转，轮流执行应用程序。

2.

1000s 时，CPU 响应中断，现运行进程 PX 用户态运行。中断优先级是 0，响应时钟中断请求。在维护 Time::lbolt 与 u_time 时关中断，暂时不响应随后到来的磁盘中断请求。由于先前为用户态，程序执行后半段的事务处理。待 STI、EOI 开中断后，时钟中断优先级降为 0，响应磁盘中断请求（中断嵌套），。

执行磁盘中断处理程序，唤醒进程 PB，RunRun++。磁盘中断处理程序执行完毕后，先前核心态不调度，不考虑剥夺 PX 进程，它继续运行时钟中断处理程序的后半段耗时的事务处理，包括唤醒进程 PA，RunRun++(值是 2)。执行完毕后，由于先前为用户态，执行例行调度。

候选进程中，{ PX，p_pri>=100；PB，p_pri = -50；PA，p_pri = 90 }。选择 PB 上台，Swtch→Sleep→Sys_read 执行 read 系统调用后半段。随后 Sys_read→Trap，PB 优先级由 -50 升至大于等于 100，RunRun++。Trap→SystemCallEntrance，RunRun 非 0，执行例行调度。

候选进程中，{ PX，p_pri>=100；PB，p_pri >= 100；PA，p_pri = 90 }。选择 PA 上台，Swtch→Sleep→Sys_Sslep 执行时钟系统调用后半段。随后 Sys_read→Sslep，PA 优先级由 90 升至大于等于 100。Trap→SystemCallEntrance，RunRun 非 0，执行例行调度。

最后，3 个用户态进程，PA、PB、PX 时间片轮转，轮流执行应用程序。