中断与调度部分的复习题

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

一、判断题

1、进程不执行系统调用就不会入睡。　×

例外：① Unix V6++中，堆栈扩展；② 0#进程睡眠等待执行换入换出操作； ③ 缺页中断。

2、现运行进程不响应中断就不会被剥夺。　√

3、现运行进程不响应中断就不会让出 CPU。　×~~√~~

0#进程睡眠等待换入换出操作并非是因为响应中断。

4、现运行进程让出 CPU 后，一定是优先级最高的进程上台运行。　×

可能是睡眠进程/在盘交换区上，没有资格运行。

5、Unix V6++系统使用的调度算法是时间片轮转调度。　×~~√~~

可剥夺的动态优先级调度算法。UnixV6++的时间片不是精确的准 1s。

6、没有中断就没有调度（现运行进程就不会让出 CPU）。　×~~√~~

0#进程睡眠等待换入换出操作，放弃 CPU 并非是因为执行系统调用（因此即使将系统调用广义地看作中断，依然错误）。

7、用户态进程，系统中至多只有一个。　√~~×~~

现运行进程执行应用程序时是系统唯一的用户态进程。就绪、阻塞全在核心态。立即返回用户态的就绪进程也要现在核心态完成恢复用户态现场的操作。

8、Unix V6++，核心态不调度。所以，如果不是入睡或终止，现运行核心态进程不会让出 CPU。　√~~×~~

总结：

一、 Unix V6++中，现运行进程放弃 CPU 是因为：**1. 被剥夺（即被抢占）**：进程返回用户态前，发现 RunRun≠0。具体而言：① 响应时钟中断发现时间片到期；② 响应时钟、外设中断唤醒了优先级更高的进程）； **2. 主动放弃 CPU 入睡 或 进程终止**：具体而言，0#进程无法完成需要执行的内核任务，如资源不足，会入睡；其他进程，则一定是因为执行了系统调用而入睡或终止。

二、 Unix V6(++)是**可剥夺的系统**，因为存在运行态->就绪态的状态变迁。即，非入睡、非终止，进程也可能放弃 CPU。

三、 Unix V6(++)的**内核不可剥夺**。因为核心态不调度，现运行进程只有在完成所有的内核

任务返回用户态或入睡、终止时才会放弃 CPU。不存在内核任务执行了一半，现运行进程被剥夺变 SRUN 的情况。

四、 常见的 Windows、Linux 系统，都是**可剥夺的系统、可剥夺的内核**。

二、 系统调用不同于一般的子程序调用。请问：UNIX V6++和 Linux 的系统调用如何

1、传递应用程序想要执行的系统调用号？ **将系统调用号存放在 EAX 寄存器中**

2、传递系统调用的参数？ **将调用的参数存放在 EBX、ECX、EDX、EDI、ESI 寄存器中，若不够用可在寄存器中存放指向内存地址的指针**

3、将系统调用的返回结果传给应用程序？ **将返回结果存放在 EAX 寄存器中，若多个返回结果也可使用 EBX 等寄存器，或是钩子函数传入的指针变量所指的用户内存区域**

三、 言简意赅

1、描述 20#系统调用的执行过程。

答：1. 应用程序通过调用系统的标准函数库中的钩子函数实行系统调用。

2.在钩子函数中，执行内联汇编语句：“__asm__ volatile ("int $0x80":"=a"(res):"a"(20)) ;
”将中断号 20 写入 EAX 寄存器，并通过 int 指令向 CPU 发送 0x80 号中断请求，激活内核系统调用陷入内核。~~CPU 响应中断后，当前进程转为核心态执行系统调用。~~

3.系统调用入口函数 SystemCallEntrance()保护用户态寄存器，将系统调用号 20 存入核心栈；

4. 系统调用处理程序 Trap()，从核心栈取出系统调用号，查系统调用表 m_SystemEntranceTable，间接调用系统调用子程序 Sys_Getpid()；

5.Sys_Getpid()将系统调用的返回值(ppid)存入核心栈，u_ar0 指向的单元。系统调用完成后，将返回值放在 EAX 寄存器中。钩子函数将其传回应用程序。~~返回到入口程序的例行调度处，选择新进程上台。当原进程再次调度上台时，取出 EAX 寄存器的值用于使用。~~

2、描述为 Unix V6++系统添加一个新的系统调用的过程。

答： 1. 在 SystemCall.cpp 中添加对新系统调用处理子程序的声明；

2. 在 SystemCall.h 文件中添加该系统调用处理子程序的声明；

3. 在 SystemCall.cpp 中找到对系统调用子程序入口表 m_SystemEntranceTable，选择系统调用号码的对应项，填入新的调用程序所需参数与入口地址的信息。

四、 请回答以下问题，言简意赅补齐系统中断响应和调度过程。

1、T0 时刻整数秒，系统中 SRUN 进程 PA 和 PB。现运行进程 PA 执行 sleep（10）系统调用。

答：1. ~~sleep 函数发出 INT 0x80 中断请求，激活内核系统调用；~~PA 执行系统调用，陷

入核心态，保护用户态现场。

2．CPU 响应中断，PA 进程进入核心态执行系统调用函数 Sys_Sslep。PA 进程设置存储 wakeTime，维护 Time::tout，并入睡等待放弃 CPU 倒计时结束；

3．由于 PA 进程入睡，Swtch 进程选中唯一的 SRUN 进程 PB 上台运行。

2、现运行进程 PA SRUN，正在执行系统调用。T1 时刻，响应中断，唤醒一个睡眠进程 PB。问，PB 进程何时上台运行？简述系统中断响应，调度过程和 PB 唤醒后上台运行。

答：1．T1 时刻，系统从系统调用程序响应中断，发生中断嵌套，并唤醒 PB；

2．中断处理程序返回后，先前系统调用为核心态，不发生例行调度，直接返回继续执行系统调用程序直至结束；

3． 系统调用程序执行结束后，先前为用户态，例行调度选择 p_pri 最低的 SRUN 进程上台运行。此时 PB 已经就绪，若 p_pri 最低就可以上台运行。

3、T2 时刻整数秒，CPU 关中断执行硬盘中断处理程序，硬盘中断处理程序的先前态是用户态。时钟中断何时响应？时钟 time 的调整是否会延迟，延迟到什么时候？

答：1．由于 CPU 关中断，无法立即响应时钟中断。时钟中断在 CPU 开中断时响应；

2．由于硬盘中断处理程序是在核心态运行，因此本次整数秒的时钟中断不会进入后半段的 Time:: time 维护阶段，直接 EOI 返回。Time::time 的调整需要延迟到下一个整数秒的时钟中断处理程序（且响应时当前进程必须在用户态）。

五、擦掉红色的判断，Unix V6++系统的钟就不走了。为什么 ？　　（这个题写着玩）

```
void Time::Clock( struct pt_regs* regs, struct pt_context* context )
{
    ……
    if( current->p_stat == Process::SRUN &&
            (context->xcs & USER_MODE) == KERNEL_MODE )
    {
        发 EOI 命令;
        return;
    }

    Time::lbolt -= HZ;
    Time::time++; //修改 wall clock time
    ……
}
```

答：若当前 CPU 处于空闲状态（idle），没有进程占用，则当前进程为零号进程，其处在核心态（KERNEL_MODE）睡眠（SSLEEP）。若不加上 current->p_stat==Process::SRUN 的判断，则许多时候响应时钟中断时系统处于上述状态中。此时时钟中断不执行后续的 Time::time 变量维护，导致 time 不能被及时更

新，因此系统的时钟便不走了。

六、Setpri( )有没有一点儿怪。说出你的疑惑，尝试解释原系统设计的合理性，或对它进行质疑。　　（这个题写着玩）

答：

```
void Process::SetPri()
{
    int priority;
    ProcessManager& procMgr = Kernel::Instance().GetProcessManager
();

    priority = this->p_cpu / 16;
    priority += ProcessManager::PUSER + this->p_nice;

    if ( priority > 255 )
    {
        priority = 255;
    }
    if ( priority > procMgr.CurPri )
    {
        procMgr.RunRun++;
    }
    this->p_pri = priority;
}
```

我的疑问是最后 priority > procMgr.CurPri 这一部分的判断。根据课本 P175，SetPri 函数主要在以下三种情况调用：① 系统调用处理程序结束时；②每秒结束时，计算现运行进程的优先数； ③每秒结束时，计算所有 p_pri 大于 PUSER 的进程优先数。

该判断的含义是“计算所得的优先数大于现运行进程被调度占用处理机时的优先数”，希望在这几次调用时尽可能的将 RunRun++，从而实现例行调度。但是在情况③中，是否应该是 priority < procMgr.CurPri 时，才说明有进程更值得在时间片轮转时上台，才更有 RunRun++的必要？

Unix 的理念是所有行为基于现运行进程。也就是说，只有现运行进程优先级下降时才考虑让出 CPU。因此，对于现运行进程，Setpri()重算优先数，RunRun++；对于非现运行进程，Setpri 仅重算优先数，RunRun 会错置，激活调度，但 select 选中的一定是优先级最高的进程。

七、你自己的任何设计或想法　　（这个题写着玩）

答：在 SetPri 中设置 RunRun 的目的是：1．在离开核心态返回用户态时，都设置调度标志； 2．每秒钟都尽量设置调度标志，鼓励进程轮转上台。

因此若想要在更合适的时机设置 RunRun，为何不直接在系统调用函数返回处以及时钟中断处理程序中，这些需要设置调度标志的地方直接对 RunRun 做设置？这样就无需在 SetPri 函数里用似乎有些奇怪的判断条件来设置调度标志了。