OS第二章课后作业

姓名	学号	院系	邮箱
张洋彬	191220169	计算机科学与技术系	1016466918@qq.com

1. **In Fig. 2-2, three process states are shown. In theory, with three states, there could be six transitions, two out of each state. However, only four transitions are shown. Are there any circumstances in which either or both of the missing transitions might occur?**

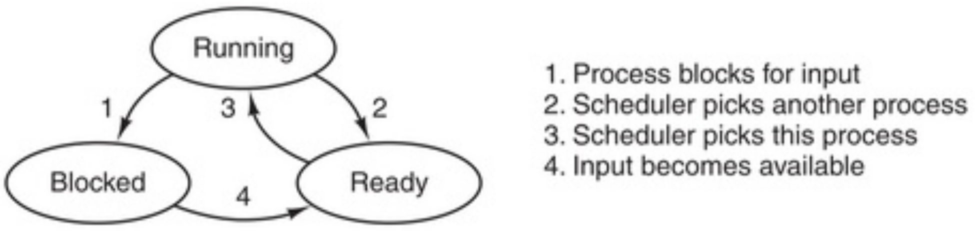

Running
Blocked
Ready
1
2
3
4
1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Figure 2-2. A process can be in running, blocked, or ready state. Transitions between these states are as shown.

从等待态直接到运行态是可能发生的，假设一个进程在I/O上阻塞，此时I/O结束，而且此时cpu空闲的话，这个进程就会直接从等待态变为运行态。而从就绪态变为等待态是不可能发生的，一个就绪进程是不可能做任何会产生阻塞的 I/O 或者别的什么事情。只有运行的进程才能被阻塞。

2. **On all current computers, at least part of the interrupt handlers are written in assembly language. Why?**

   首先是开中断和关中断，用c语言无法表达，后来会将现场信息保存在核心栈中以及设置堆栈指针，这些操作无法用C语言这一类高级语言描述，所以中断的部分内容需要汇编语言例程来完成。

3. **When an interrupt or a system call transfers control to the operating system, a kernel stack area separate from the stack of the interrupted process is generally used. Why?**

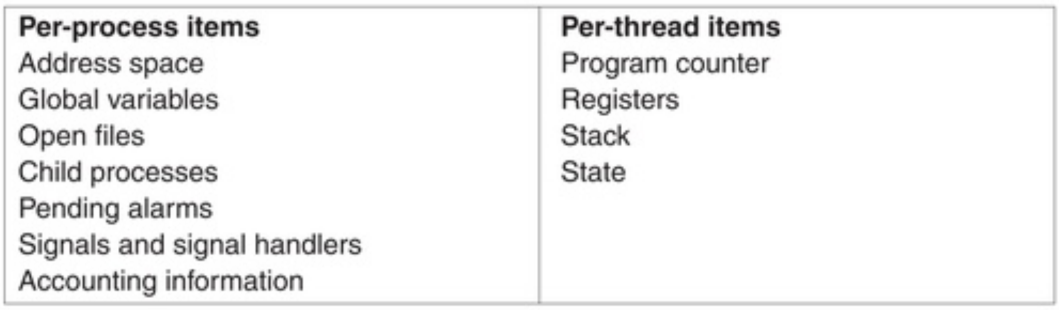
   当从用户态进入内核态，或是从内核态返回用户态时，会发生堆栈的变化。从用户态进入内核态时，是从特权级别为3切换到特权级为0，首先需要从TSS中获取ss0和esp0，因为进入到了内核态后还会返回用户态，所以需要将当前的ss和esp压入内核栈（所以进入内核态要切换成内核栈），随后把标志积存器eflags的内容和返回位置cs，eip压入内核态堆栈。当从内核态返回用户态时，要从内核栈里弹出这些值，恢复陷入中断之前的状态。而且用户栈无法存储内核的一些操作的数据，所以必须使用内核栈。

4. **A computer has 4 GB of RAM of which the operating system occupies 512 MB. The processes are all 256 MB (for simplicity) and have the same characteristics. If the goal is 99% CPU utilization, what is the maximum I/O wait that can be tolerated?**

   空闲的RAM = 4096 MB - 512MB = 3584 MB
   可并发的进程数量 3584 / 256 = 14
   $1 - p^{14} = 0.99$
   $1 - 0.99 = p^{14}$
   p=root14(0.01)=71.97%

   最大的I/O等待时间是71.97%

5. **In Fig. 2-12 the register set is listed as a per-thread rather than a per-process item. Why? After all, the machine has only one set of registers.**



Per-process items	Per-thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

Figure 2-12. The first column lists some items shared by all threads in a process. The second one lists some items private to each thread.

   因为一个进程内的多个线程共享进程所获得的内存空间与资源，为完成某一项任务而协同工作。因为多个线程在同一个进程中运行或等待，他们共享的内存空间与资源在运行阶段是相互独立的（一套寄存器被“复制”成了多套寄存器），所以进程封装的是管理信息，包括对指令代码、全局数据、打开的文件和信号量等共享部分的管理，线程封装的是执行信息，包括状态信息、寄存器、执行栈和局部变量、过程调用参数、返回值等私有部分的管理。所以寄存器会被列到线程的部分里面。当线程停止时，它在寄存器中具有值。所以必须保存它们，就像进程停止时一样，必须保存寄存器的值。多线程与进程没有什么不同，因此每个线程都需要有自己的寄存器保存区。

6. **Why would a thread ever voluntarily give up the CPU by calling thread yield? After all, since there is no periodic clock interrupt, it may never get the CPU back.**

   在一个进程里的线程是合作关系，当为了更好的运行时，当前运行的线程就会调用thread_yield()，从运行态转为就绪态，转而让其他线程占用cpu运行。在后续，它可能会转回运行态，也可能会一直在就绪态。毕竟所有线程都是由一个程序员写的，是为了更好的运行才会让正在运行的线程变为就绪态。

7. **What is the biggest advantage of implementing threads in user space? What is the biggest disadvantage?**

   最大的优点是效率，线程的切换无需使用内核特权方式，所有线程管理的数据结构都在用户空间中，可以节省模式转换开销和内核的宝贵资源。最大的缺点是一个用户级线程的阻塞将会引起整个进程阻塞。

8. **Can a measure of whether a process is likely to be CPU bound or I/O bound be determined by analyzing source code? How can this be determined at run time?**

   在简单情况下，可以通过分析源代码看出进程是否是I/O密集型，比如，一个程序将其所有输入文件读入缓冲区中，开始可能不是I/O密集型，即cpu的loading很高，但是随着读写的增多，CPU需要等待I/O (硬盘/内存) 的读/写操作，此时就会是I/O密集型。如果操作系统提供了诸如UNIX ps命令之类的功能，可以得到程序使用的CPU时间量的话，就可以将程序所消耗的时间和执行这个程序所消耗的总时间进行比较，就可以判断是CPU密集型还是I/O密集型。

9. **Consider a real-time system with two voice calls of periodicity 5 msec each with CPU time per call of 1 msec, and one video stream of periodicity 33 ms with CPU time per call of 11 msec. Is this system schedulable?**

   将所有的周期性事件的cpu占用时间/总时间的比值加起来，如果小于1，则说明是可调度的。

   1/5+1/5+11/33=11/15<1,说明这个系统是可调度的。

10. **Consider the following piece of C code: void main( ) { fork( ); fork( ); exit( ); } How many child processes are created upon execution of this program?**

    2^2-1=3，有三个子进程。

11. **Five batch jobs. A through E, arrive at a computer center at almost the same time. They have estimated running times of 10, 6, 2, 4, and 8 minutes. Their (externally determined) priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead. (a) Round robin. (b) Priority scheduling. (c) First-come, first-served (run in order 10, 6, 2, 4, 8). (d) Shortest job first. For (a), assume that the system is multiprogrammed, and that each job gets its fair share of the CPU. For (b) through (d), assume that only one job at a time runs, until it finishes. All jobs are completely CPU bound.**

    a）Round robin（假设时间片为1）

    顺序ABCDEABCDEABDEABDEABEABEAEAEAA

    t(A) = 30, t(B) = 23, t(C) = 8, t(D) = 17, t(E) = 28, average = 21.2（minutes）

    b）Priority scheduling

    顺序：BEACD

    t(A) =24 , t(B) = 6, t(C) = 26, t(D) = 30， t(E) = 14, average = 20（minutes）

    c）First-come, first-served (run in order 10, 6, 2, 4, 8).

    顺序：ABCDE

    t(A) = 10, t(B) = 16, t(C) = 18, t(D) = 22, t(E) = 30, average = 19.2（minutes）

    d）Shortest job first

    顺序：CDBEA

    t(A) = 30, t(B) = 12, t(C) = 2, t(D) = 6, t(E) = 20, average = 14（minutes）

12. **The aging algorithm with a = 1/2 is being used to predict run times. The previous four runs, from oldest to most recent, are 40, 20, 40, and 15 msec. What is the prediction of the next time?**

$Tn+1=a*tn+(1-a)Tn$

$Tn+1=tn/2+Tn/2$

$T1=t0/2+T0/2$

$T2=t1/2+T1/2$

$=t1/2+(t0/2+T0/2)/2$

$=t1/2+t0/4+T0/4$

$T3=t2/2+T2/2$

$=t2/2+(t1/2+t0/4+T0/4)/2$

$=t2/2+t1/4+t0/8+T0/8$

$T4=t3/2+T3/2$

$=t3/2+(t2/2+t1/4+t0/8+T0/8)/2$

$=t3/2+t2/4+t1/8+t0/16+T0/16$

$T0=t0=40$

$t1=20$

$t2=20$

$t3=15$

$T4=t3/2+t2/4+t1/8+t0/16+T0/16T4$

$=15/2+20/4+20/8+40/16+40/16$

$=7.5+5+2.5+2.5+2.5$

$=20$ msec