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 are shown. Are there circumstances in which either or both of the missing transitions might occur?
就绪态不会直接转变为阻塞态（等待态）。进程只有在运行态出现等待事件时，才会主动发出请求，进入等待态。等待态也不会直接变为运行态。等待事件发生后，进程必须先变成就绪态，然后才能由处理器调度，进入运行态。

2. On all current computers, at least part of the interrupt handlers is written in assembly language. Why?
通常情况下，高级语言是没有权限访问 CPU 硬件的。中断处理程序过程中必然涉及到对 CPU 的访问，因此这部分工作必须用汇编语言来实现。此外，中断处理程序对运行速度有较高要求，因此使用汇编语言直接操作比高级语言更适合。

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?
首先，如果将内核的数据和被打断用户进程的堆栈放在一起，则用户程序有可能访问和修改内核数据，造成不安全的行为。其次，用户程序的堆栈大小有限，如果在其上直接开辟空间存放内核数据，有可能空间不够而造成堆栈溢出。

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?
RAM 可容纳的进程数量为(4GB – 512MB) / 256MB = 14 个，设每个进程的 I/O 等待时间占总运行时间的比例为 p，则有 1 – p^14 >= 99%，解得 p <= 71.9%。

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.
每个线程的执行是相对独立的，需要单独管理和维护各自的寄存器和状态信息，在线程结束时才保存到进程的空间中。因此寄存器是线程的私有内容。

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.
进程中的线程是合作而非竞争关系，有时为了程序整体更好地运行，个别线程需要暂时主动让出处理器，从而让其他线程能够使用 CPU，配合完成工作。

7. What is the biggest advantage of implementing threads in user space? What is the biggest disadvantage?
优点：线程切换不涉及模式切换（代价小），调度算法的选择灵活；
缺点：不能同时在多个处理器上运行，一个线程的阻塞将导致整个进程的阻塞。

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?
有时可以通过分析源代码来判断进程是 CPU 密集型还是 I/O 密集型的。
在运行过程中，如果程序一开始就把用到的大部分文件读入缓冲区，然后再集中进

行操作，操作完毕后统一回写，那么程序是 CPU 密集型的；如果程序一边运行一边频繁地对文件进行读写，那么程序是 I/O 密集型的。

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?
(1 / 5) * 2 + 11 / 33 = 2 / 5 + 1 / 3 = 11 / 15 < 1，因此系统是可调度的。

10. Consider the following piece of C code: void main( ) { fork( ); fork( ); exit( ); } How many child processes are created upon execution of this program?
三个，代码运行时，第一个 fork 由父进程创建了一个子进程，第二个 fork 由父进程和子进程各自又创建了一个子进程，因此共创建了三个子进程。

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) 按照每个时间片 2 分钟轮转运行，顺序为 ABCDEABDEABEAEA，其中 A 从到达 CPU 到执行完毕经过 10 + 8 + 6 + 4 + 2 = 30 分钟，B 为 10 + 8 + 4 = 22 分钟，C 为 6 分钟，D 为 10 + 6 = 16 分钟，E 为 10 + 8 + 6 + 4 = 28 分钟，则平均进程周转时间应为(30 + 22 + 6 + 16 + 28) / 5 = 20.4 分钟
    b) 按照优先级执行，顺序为 BEACD，其中 A 花费 6 + 8 + 10 = 24 分钟，B 为 6 分钟，C 为 24 + 2 = 26 分钟，D 为 26 + 4 = 30 分钟，E 为 6 + 8 = 14 分钟，则平均进程周转时间应为(24 + 6 + 26 + 30 + 14) / 5 = 20 分钟
    c) 按照到达 CPU 的时间依次执行，顺序为 ABCDE，其中 A 花费 10 分钟，B 为 10 + 6 = 16 分钟，C 为 16 + 2 = 18 分钟，D 为 18 + 4 = 22 分钟，E 为 22 + 8 = 30 分钟，则平均进程周转时间为(10 + 16 + 18 + 22 + 30) / 5 = 19.2 分钟
    d) 按照进程运行所需的时间长短从小到大依次执行，顺序为 CDBEA，其中 A 花费了 30 分钟，B 花费 12 分钟，C 花费 2 分钟，D 花费 6 分钟，E 花费 20 分钟，平均进程周转时间为(30 + 12 + 2 + 6 + 20) / 5 = 14 分钟

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?
四次运行过程中，得到的预测值依次为：40 msec —> (40 + 20) / 2 = 30 msec —> (30 + 40) / 2 = 35 msec —> (35 + 15) / 2 = 25 msec，即下一次的预测值为 15ms。