## 评分标准

满分100分。

选择题每题6分（共10道选择题），大题评分标准在参考答案中给出（8分+16分+16分）。

## 参考答案

### Part1 死锁

1. 在操作系统中，死锁出现是指( A )

A.多个进程竞争资源出现了循环等待 B. 一个进程进入死循环

C. 进程释放资源 D. 多个进程竞争使用共享型的设备

2. 死锁的四个必要条件中，无法破坏的是( C )

A．非抢夺式分配 B．占有且等待资源

C．互斥使用资源 D．环路等待资源

3. 若系统 S1采用死锁避免方法，S2采用死锁检测方法。下列叙述中，正确的是( B )

I. S1会限制用户申请资源的顺序，而 S2 不会

II. S1需要进程运行所需的资源总量信息，而 S2不需要

III. S1不会给可能导致死锁的进程分配资源，而 S2会

A．仅 I、II B．仅 II、III C．仅 I、III D．I、II、III

解析：

1. 死锁预防采用破坏产生死锁的4个必要条件中的一个或几个来防止发生死锁。其中之一的“破坏循环等待条件”，一般采用资源有序分配法，限制用户申请资源的顺序，因此**I的前半句属于死锁预防的范畴**。因此I错误。

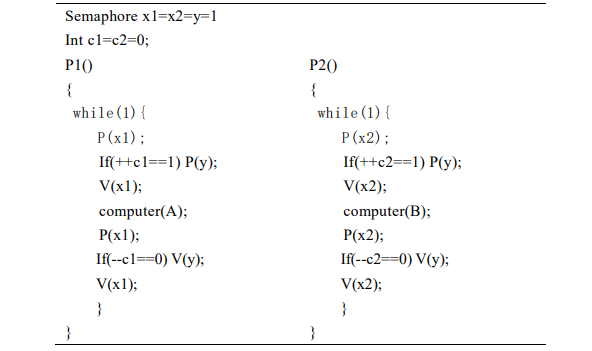
2. 银行家算法是最著名的死锁避免算法，需要资源总量信息，若检测到不安全，则本次分配作废。而在死锁检测中，系统为进程分配资源时不采取任何措施，但提供死锁的检测手段。因此II和III正确。

4、某系统有n台互斥使用的同类设备，三个并发进程分别需要3，4，5台设备，可确保系统不发生死锁的设备数n最小为( B )

A. 9 B. 10 C. 11 D. 12

解析：考虑极端情况，3个进程分别申请到2/3/4台设备，此时产生死锁。只要再增加一台设备就能避免以上情况，因此n最小为2+3+4+1=10。

5. 下面是一个并发进程的程序代码，正确的是( B )



A. 进程不会死锁，也不会“饥饿” B. 进程不会死锁，但是会“饥饿”

C. 进程会死锁，但是不会“饥饿” D. 进程会死锁，也会“饥饿”

解析：若进程执行过程如下：①假设 P1进程稍快，P2进程稍慢，同时运行。②P1进程首先进入 if 条件语句，因此获得了 y 的互斥访问权，P2被阻塞；③在第一个 P1进程未释放 y 之前，又有另一个 P1进入，c1 的值变成 2，当第一个 P1 离开时，P2仍然被 阻塞，这种情形不断发生；④在这种情况下，P1顺利执行，P2长期被阻塞。

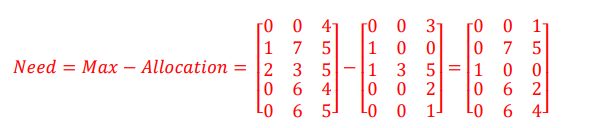
6.

评分标准：满分8分。并发性排序正确给满分；若并发性排序错误，但答出“死锁检测方法”并发性最高并说明理由给4分。

解析：死锁检测方法可以获得最大的并发性。并发性排序从小到大排序：资源预分配、银行家算法、死锁检测方法。

7.

评分标准：满分16分。Need矩阵4分，正确分析每种情况3分。



解析：

I. 根据 Need 矩阵可知，当 Available 为(1,4,0)时，可以满足 P2 的需求；P2结束释放资源之后，Available为(2,7,5)可以满足 P0、P1、P3、P4中任一进程的需求，所以系统不会出现死锁，处于安全状态。

II. 当 Available 为(0,6,2)时，系统可以满足进程 P0、P3的需求，二者释放资源 后，Available 为(0,6,7)，仅可以满足 P4的需求。P4结束释放资源后，Available 为(0,6,8)，此时不能满足余下任一进程的需求，系统出现死锁，当前处于非安全状态。

III. 当 Available 为(1,1,1)时，系统可以满足 P0、P2的需求；这两个进程结束后释放资源，Available 为(2,4,9)，此时不能满足余下任一进程的需求，系统出现死锁，当前处于非安全状态。

IV. 当 Available 为(0,4,7)时，系统可以满足 P0的需求；进程结束后释放资源， Available 为(0,4,10)，此时不能满足余下任一进程的需求，系统出现死锁，当前处于非安全状态。

### Part2 CPU调度

1. 若某单处理器多进程系统中有多个就绪态进程，则下列关于处理机调度的叙述中，错误的是( D )。

A. 创建新进程后能进行处理机调度

B. 在进程结束时能进行处理机调度

C. 在系统调用完成并返回用户态时能进行处理机调度

D. 在进程处于临界区时不能进行处理机调度

解析：当进程处于临界区时，说明进程正在占用处理机，只要不破坏临界资源的使用规则，是不会影响处理机调度的。

1. P1、P2和P3是在某个系统中正在执行的三个进程，各进程的计算(CPU)时间和I/O时间比例如下表所示：

|  |  |  |
| --- | --- | --- |
| 进程 | 计算时间 | I/O 时间 |
| P1 | 50% | 50% |
| P2 | 20% | 80% |
| P3 | 30% | 70% |

为提高系统资源利用率，合理的进程优先级设置是( C)。

1. P1>P2>P3 B. P3>P2>P1 C. P2>P3>P1 D. P1>P3>P2

解析：计算进程会占用大量的cpu时间，而i/o大的会占用较少的cpu资源，相当于短作业，所以应该优先权更高。

1. 下列选项中，降低进程优先级的合理时机是(B)。

A.进程刚完成I/O，进入就绪队列 B. 进程的时间片用完

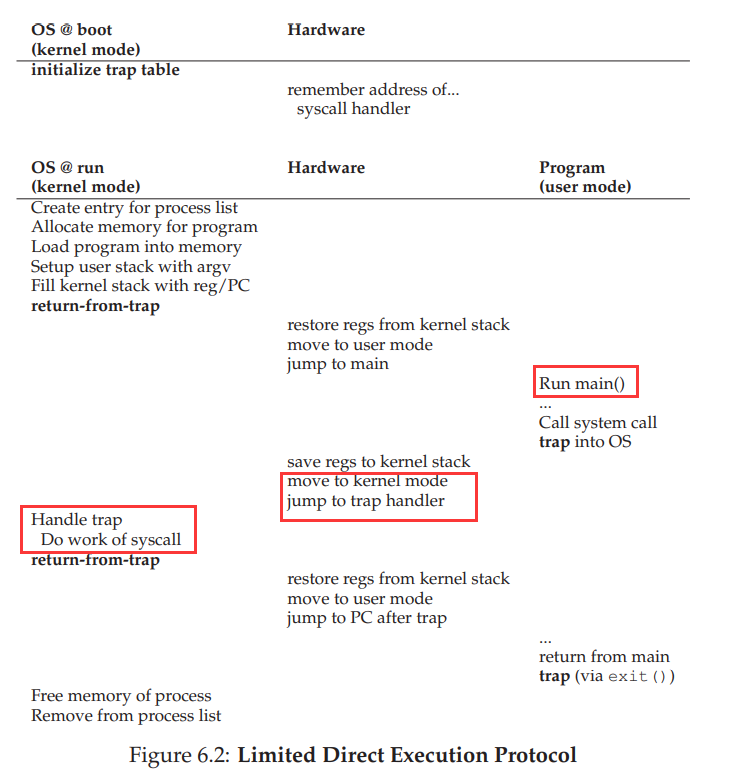
C.进程从就绪状态转为运行态 D. 进程长期处于就绪队列中

解析：对于A选项，该进程已经进入就绪态；对于C选项，该进程刚运行就降低优先级，可能被抢断；对于D选项，该进程长期未能运行，再降低优先级可能饿死。

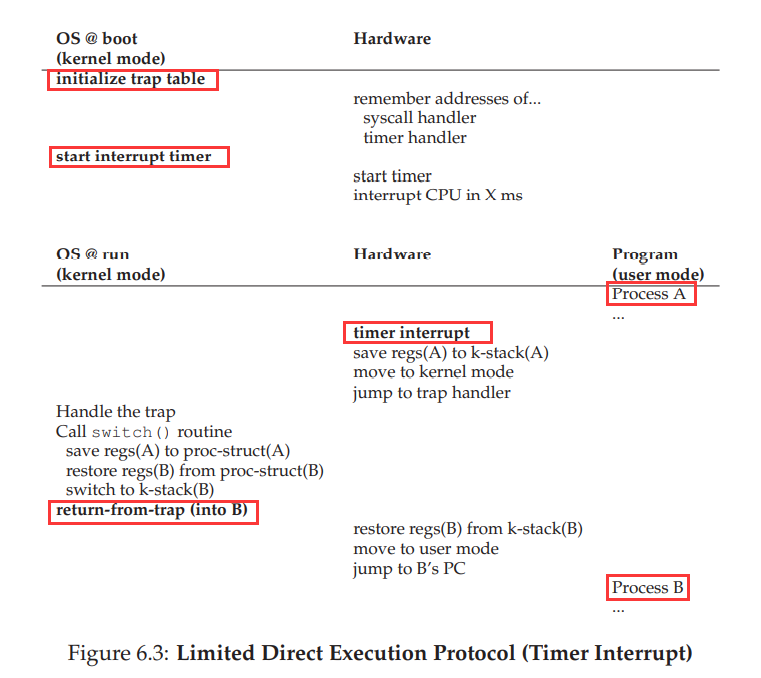
1. 陷阱指令（trap）可以使执行流程从用户态陷入内核，在用户进程使用陷阱（trap）执行调用内核函数的过程中，以下哪些步骤是由操作系统内核完成的( A )  
    ① 执行用户进程的main函数  
    ② 切换至内核模式  
    ③ 跳转至陷阱处理器（trap handler）  
    ④ 处理陷阱（handle trap）  
    ⑤ 执行系统调用

A.④⑤ B.③④⑤ C.②③④⑤ D.①⑤

解析：题目参考了《Operating Systems-Three Easy Pieces》 第6章第5页的Figure 6.2。系统调用(system call)是提供给用户程序用于执行更高权限操作的途径。为了执行系统调用，程序必须先执行特殊的trap指令，硬件将特权等级切换至内核模式；系统调用结束后，操作系统会调用特殊的return-from-trap指令，返回回用户程序。但为了保证能正确地返回，硬件需要在执行trap指令时保存调用者（用户程序）的一些寄存器，并在操作系统处理return-from-trap时取出。因此，只有处理trap和执行系统调用是由操作系统完成的。



5. 时钟中断（timer interrupt）是整个操作系统的脉搏，系统利用时钟中断维持系统时间、**保证进程共享CPU**、确定调度优先级。考虑如下场景：系统开机后，先执行进程A，接着遇到时钟中断，转为执行进程B，过程中需要执行以下操作  
 ① 从陷阱中返回（return from trap）  
 ② 初始化陷阱表（initialize trap table）  
 ③ 处理时钟中断（timer interrupt）  
 ④ 开启中断计时器（start interrupt timer）  
 ⑤ 执行进程A  
 ⑥ 执行进程B  
操作的正确顺序是( D )  
A.②④⑤③⑥① B. ⑤②④③⑥① C.④②⑤③①⑥ D.②④⑤③①⑥  
解析：题目参考了《Operating Systems-Three Easy Pieces》 第6章第10页的Figure 6.3。初始化陷阱表（initialize trap table），让操作系统告诉硬件trap table放在内存中的什么位置，一般在系统启动时就会执行这条指令，因此初始化陷阱表是最先执行的。为了使得不同的进程能在CPU上切换，CPU需要隔一段时间就暂停执行当前进程，转去执行别的进程，这就需要时钟中断（timer interrupt）。因此在系统启动后，需要开启中断计时器，记录系统执行当前进程的时间，这是第二个操作。接着系统执行进程A，这是第三个操作。在执行A的过程中，中断计时器时间结束，触发了时钟中断（timer interrupt），这是第四个操作。此时系统trap到内核模式，中断结束后通过return-from-trap从内核模式返回，这是第五个操作。返回后，CPU开始执行进程B。因此正确的执行顺序是②④⑤③⑥①。



6.

评分标准：

满分16分。

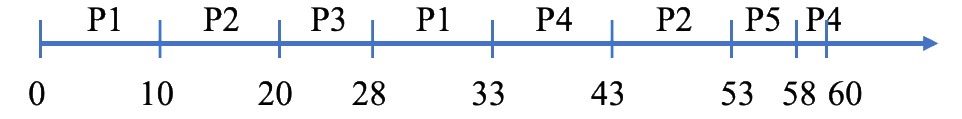
问题（1）6分，调度结果图正确给满分，部分正确可酌情给分。

问题（2）4分，调度结果图正确给满分，部分正确可酌情给分。

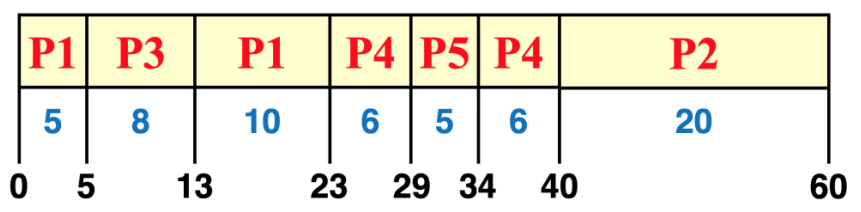
问题（3）6分，每个时间计算正确给1.5分。

解析：

1）



2）



3）

平均响应时间:

RR：( 0+10+(20-5)+(33-13)+(53-29) ) / 5 = 13.8 ms

SJF：( 0+40+(5-5)+(23-13)+(29-29) ) / 5 = 10 ms

平均周转时间

RR：( (33-0)+(53-0)+(28-5)+(60-13)+(58-29) ) / 5 = 37 ms

SJF：( (23-0)+(60-0)+(13-5)+(40-13)+(34-29) ) / 5 = 24.6 ms