1. 关于处理器调度，试问:

   1)什么是处理器的三级调度?

   2)处理器的三级调度分别在什么情况下发生?

   3)各级调度分别完成什么工作?

1）处理器的三级调度是指一个作业在运行过程中要遇到的高级调度（作业调度）、中级调度（进程对换）和低级调度（进程调度）。不过，不是所有的操作系统都有三级调度，有些只实现了其中的一级或两级，但是每个操作系统都有进程调度。

2）高级调度主要在需要从外存调入一个作业到内存中时发生；中级调度主要在内存紧张需要调出一些进程，或者内存空闲需要把先前调出的进程调回内存时发生；低级调度主要在正在执行的进程放弃 CPU 或者被其他优先级高的进程抢占 CPU 时发生。

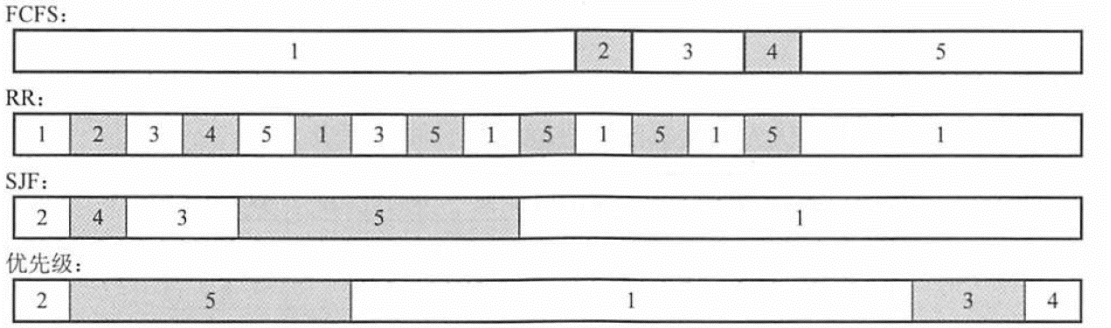
3）高级调度的主要工作是决定外存的后备队列中哪个进程被调入到内存中，并给这个作业创建进程，给它分配必要的资源；中级调度的主要工作是在内存紧张时把就绪队列中暂时得不到执行的进程换到外存，也负责在内存较空闲时把换到外存的进程调回内存；低级调度的主要工作是决定把 CPU 分配给就绪队列中的哪个进程。

2. 假定要在一台处理器上执行下表中的作业，且假定这些作业在时刻 0 以 1~5 的顺序到达(数字越小，优先级越高)。说明分别使用 FCFS（先来先服务）、

| 作　业 | 执行时间/s | 优 先 级 |
|---|---|---|
| 1 | 10 | 3 |
| 2 | 1 | 1 |
| 3 | 2 | 3 |
| 4 | 1 | 4 |
| 5 | 5 | 2 |

RR(时间片轮转，时间片=1s)、SJF（短作业优先）以及非抢占式优先级调度算法时，这些作业的执行情况。针对上述每种调度算法，列出包括到达时间、服务时间、开始时间、完成时间、周转时间和带权周转时间的表格，并给出平均周转时间和平均带权周转时间。

作业执行情况可以用如下的甘特图来表示。




各个作业对应于各个算法的周转时间和加权周转时间见下表。

| 算法 | 时间类型 | $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ | 平均时间 |
|---|---|---|---|---|---|---|---|
| | 运行时间 | 10 | 1 | 2 | 1 | 5 | 3.8 |
| FCFS | 周转时间 | 10 | 11 | 13 | 14 | 19 | 13.4 |
| | 加权周转时间 | 1 | 11 | 6.5 | 14 | 3.8 | 7.26 |
| RR | 周转时间 | 19 | 2 | 7 | 4 | 14 | 9.2 |
| | 加权周转时间 | 1.9 | 2 | 3.5 | 4 | 2.8 | 2.84 |
| SJF | 周转时间 | 19 | 1 | 4 | 2 | 9 | 7 |
| | 加权周转时间 | 1.9 | 1 | 2 | 2 | 1.8 | 1.74 |
| 优先级 | 周转时间 | 16 | 1 | 18 | 19 | 6 | 12 |
| | 加权周转时间 | 1.6 | 1 | 9 | 19 | 1.2 | 6.36 |

**所以，FCFS 的平均周转时间为 13.4，平均加权周转时间为 7.26。**
**RR 的平均周转时间为 9.2，平均加权周转时间为 2.84。**
**SJF 的平均周转时间为 7，平均加权周转时间为 1.74。**
**非剥夺式优先级调度算法的平均周转时间为 12，平均加权周转时间为 6.36。**

**注意：SJF 的平均周转时间肯定是最短的，计算完毕后可以利用这个性质进行检验。**

3. 某系统有同类资源 m 个，供 n 个进程共享。假设每个进程最多申请 x 个资源(其中 $1 \le x \le m$)，请证明:当 $n(x-1)+1 \le m$ 时，系统不会发生死锁。

【解析】由于每个进程最多申请使用 x 个资源，在最坏的情况下，每个进程都得到了 x-1 个资源，并且现在均需申请所需的最后一个资源。这时系统剩余资源数为 m-n(x-1)。如果系统的剩余资源数大于或等于 1，即系统至少还有一个资源可以使用，就可以使这 n 个进程中的任一个进程获得所需的全部资源。该进程可以运行结束，释放出所占有的资源，供其他进程使用，从而使每个进程都可以执行结束。

因此，当 m-n(x-1)≥1 时，即 n(x-1)+1≤m 时，系统不会发生死锁。

本题需要对资源分配的最坏情况有所了解，即所有共享资源的进程都获得比最大需求资源量少一个的资源，并都需要申请所需的最后一个资源。

根据本题，可以得出推论：n(x-1)+1>m 时，系统可能发生死锁。该推论可以作为判断系统是否发生死锁的通用方法。该结论及推论的使用需要建立在理解的基础上，如果只是死记硬背，容易出错。

4. 在银行家算法中，出现以下资源分配情况(见下表)。系统剩余资源数量=(3，2，2)

| 进　　程 | 资源最大需求 | 已分配资源 |
|---|---|---|
| $P_0$ | 7，5，3 | 0，1，0 |
| $P_1$ | 3，2，2 | 2，1，0 |
| $P_2$ | 9，0，2 | 3，0，2 |
| $P_3$ | 2，2，2 | 2，1，1 |
| $P_4$ | 4，3，3 | 0，0，2 |

试问:

1)该状态是否安全?请给出详细的检查过程。

2)若进程依次有如下资源请求:

P1:资源请求 Request(1，0，2)

P4:资源请求 Request(3，3，0)

P0:资源请求 Request(0，1，0):

则系统该如何进行资源分配，才能避免死锁?

1）系统安全，因为存在安全序列（$P_1$，$P_3$，$P_0$，$P_2$，$P_4$）。过程如下：

先求出各进程剩余的需求量：

$P_0$=（7，4，3），$P_1$=（1，1，2），$P_2$=（6，0，0），$P_3$=（0，1，1），$P_4$=（4，3，1）。

根据系统剩余资源数（3，2，2）可知，可以立即满足的进程是 $P_1$（或 $P_3$），$P_1$ 满足后可释放占有的资源，系统剩余资源数为（5，3，2），找到可立即满足的进程是 $P_3$（或 $P_4$），$P_3$ 满足后释放占有的资源，系统此时剩余资源数为（7，4，3），找到可立即满足的进程是 $P_0$（或 $P_2$、$P_4$），所有进程可以依次执行完毕。

2）系统要想避免死锁，就必须保证每次分配完后都能得到安全序列，否则就拒绝分配。根据这一原则，对于进程的请求应考虑分配以后是否安全，若不安全，则不能进行此次分配。题目中有 3 个请求，按照顺序来依次考虑。先考虑能否满足 $P_1$，分配后系统处于安全状态，因为分配后可以找到安全序列（$P_1$，$P_3$，$P_2$，$P_0$，$P_4$）。满足 $P_1$ 的请求之后，剩余资源为（2，2，0）。对于 $P_4$ 的请求，由于系统没有那么多剩余资源，因此无法满足，系统拒绝 $P_4$ 的请求。最后考虑 $P_0$ 的请求，如果满足 $P_0$，分配后剩余资源为（2，1，0），可以找到安全序列（$P_1$，$P_3$，$P_0$，$P_2$，$P_4$），因此可以满足 $P_0$ 的请求。总之，系统对 3 个请求依次处理为：满足 $P_1$、拒绝 $P_4$、满足 $P_0$。

5. 设系统中有 3 种类型的资源 A、B、C 和 5 个进程 P1、P2、P3、P4、P5，A 资源的数量为 17，B 资源的数量为 5，C 资源的数量为 20。在 T0 时刻，系统状态见下表。系统采用银行家算法实现死锁避免。

| 进程 | 资源 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 最大资源需求量 | | | 已分配资源数量 | | | 剩余资源数量 | | |
| | A | B | C | A | B | C | A | B | C |
| $P_1$ | 5 | 5 | 9 | 2 | 1 | 2 | 2 | 3 | 3 |
| $P_2$ | 5 | 3 | 6 | 4 | 0 | 2 | | | |
| $P_3$ | 4 | 0 | 11 | 4 | 0 | 5 | | | |
| $P_4$ | 4 | 2 | 5 | 2 | 0 | 4 | | | |
| $P_5$ | 4 | 2 | 4 | 3 | 1 | 4 | | | |

试问:

1)T0 时刻是否为安全状态?若是，请给出安全列。

2)在 T0 时刻，若进程 P2 请求资源(0，3，4)，是否能实施资源分配?为什么?

3)在 2)的基础上，若进程 P4 请求资源(2，0，1)，是否能实施资源分配?

4)在 3)的基础上，若进程 P1 请求资源(0，2，0)，是否能实施资源分配?

先根据 Allocation 矩阵和 Max 矩阵计算 Need 矩阵。

$$\text{Need=Max-Allocation=}\begin{pmatrix}5&5&9\\5&3&6\\4&0&11\\4&2&5\\4&2&4\end{pmatrix}-\begin{pmatrix}2&1&2\\4&0&2\\4&0&5\\2&0&4\\3&1&4\end{pmatrix}=\begin{pmatrix}3&4&7\\1&3&4\\0&0&6\\2&2&1\\1&1&0\end{pmatrix}$$

1）现在对 $T_0$ 时刻的状态进行安全性分析：

由于 Available 向量为（2，3，3），因此 Work 向量初始化为（2，3，3）。

因存在安全序列（$P_4$，$P_2$，$P_3$，$P_5$，$P_1$），见表 2-21，所以 $T_0$ 时刻处于安全状态。

**表 2-21　$T_0$ 时刻安全性检查表**

| 进程 | 矩阵 | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Work | | | Need | | | Allocation | | | Work + Allocation | | | Finish |
| | A | B | C | A | B | C | A | B | C | A | B | C | |
| $P_4$ | 2 | 3 | 3 | 2 | 2 | 1 | 2 | 0 | 4 | 4 | 3 | 7 | True |
| $P_2$ | 4 | 3 | 7 | 1 | 3 | 4 | 4 | 0 | 2 | 8 | 3 | 9 | True |
| $P_3$ | 8 | 3 | 9 | 0 | 0 | 6 | 4 | 0 | 5 | 12 | 3 | 14 | True |
| $P_5$ | 12 | 3 | 14 | 1 | 1 | 0 | 3 | 1 | 4 | 15 | 4 | 18 | True |
| $P_1$ | 15 | 4 | 18 | 3 | 4 | 7 | 2 | 1 | 2 | 17 | 5 | 20 | True |

2）$T_0$ 时刻由于 Available 向量为（2，3，3），而 $P_2$ 请求资源 $Request_2$ 向量为（0，3，4），显然 C 资源的数量不够，所以不能实施资源分配。

3）$Request_4$（2，0，1）<$Need_4$（2，2，1），$Request_4$（2，0，1）<Available（2，3，3）。

所以先试着把 $P_4$ 申请的资源分配给它，Available 变为（0，3，2）。

★注：这里别忘记修改系统的 Available 向量和 $P_4$ 的 $Allocation_4$ 以及 $Need_4$ 向量。

得到的系统状态情况见表 2-22。

**表 2-22　分配资源给 $P_4$ 进程后的系统状态情况表**

| 进程 | 矩阵 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Max | | | Allocation | | | Need | | | Available | | |
| | A | B | C | A | B | C | A | B | C | A | B | C |
| $P_1$ | 5 | 5 | 9 | 2 | 1 | 2 | 3 | 4 | 7 | | | |
| $P_2$ | 5 | 3 | 6 | 4 | 0 | 2 | 1 | 3 | 4 | | | |
| $P_3$ | 4 | 0 | 11 | 4 | 0 | 5 | 0 | 0 | 6 | 0 | 3 | 2 |
| $P_4$ | 4 | 2 | 5 | 4 | 0 | 5 | 0 | 2 | 0 | | | |
| $P_5$ | 4 | 2 | 4 | 3 | 1 | 4 | 1 | 1 | 0 | | | |

然后进行系统安全性检测：

此时 Available 为（0，3，2），所以 Work 初始化为（0，3，2）。

尝试把资源分配给 $P_4$ 后的系统安全性检测过程：

因存在安全序列（$P_4$，$P_2$，$P_3$，$P_5$，$P_1$），见表 2-23，所以系统仍处于安全状态，即能将资源分配给 $P_4$。

**表 2-23　分配资源给 $P_4$ 进程后的系统安全性检测表**

| 进程 | 矩阵 | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Work | | | Need | | | Allocation | | | Work + Allocation | | | Finish |
| | A | B | C | A | B | C | A | B | C | A | B | C | |
| $P_4$ | 0 | 3 | 2 | 0 | 2 | 0 | 4 | 0 | 5 | 4 | 3 | 7 | True |
| $P_2$ | 4 | 3 | 7 | 1 | 3 | 4 | 4 | 0 | 2 | 8 | 3 | 9 | True |
| $P_3$ | 8 | 3 | 9 | 0 | 0 | 6 | 4 | 0 | 5 | 12 | 3 | 14 | True |
| $P_5$ | 12 | 3 | 14 | 1 | 1 | 0 | 3 | 1 | 4 | 15 | 4 | 18 | True |
| $P_1$ | 15 | 4 | 18 | 3 | 4 | 7 | 2 | 1 | 2 | 17 | 5 | 20 | True |

4）$Request_1$（0，2，0）<Available（0，3，2），$Request_1$（0，2，0）<$Need_1$（3，4，7）。

所以先试着把 $P_1$ 申请的资源分配给它，Available 变为（0，1，2）。

★注：这里别忘记修改系统的 Available 向量和 $P_1$ 的 $Allocation_1$ 以及 $Need_1$ 向量。

得到的系统状态见表 2-24。

**表 2-24　分配资源给 $P_1$ 进程后的系统状态情况表**

| 进程 | 矩阵 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Max | | | Allocation | | | Need | | | Available | | |
| | A | B | C | A | B | C | A | B | C | A | B | C |
| $P_1$ | 5 | 5 | 9 | 2 | 3 | 2 | 3 | 2 | 7 | | | |
| $P_2$ | 5 | 3 | 6 | 4 | 0 | 2 | 1 | 3 | 4 | | | |
| $P_3$ | 4 | 0 | 11 | 4 | 0 | 5 | 0 | 0 | 6 | 0 | 1 | 2 |
| $P_4$ | 4 | 2 | 5 | 4 | 0 | 5 | 0 | 2 | 0 | | | |
| $P_5$ | 4 | 2 | 4 | 3 | 1 | 4 | 1 | 1 | 0 | | | |

然后进行系统安全性检测：

此时 Available 为（0，1，2），所以 Work 初始化为（0，1，2）。

此时的 Work 小于任意的 $Need_i$ 向量，而且存在 Finish[i]=false（其实是所有 Finish[i]都是 false），所以系统处于不安全状态，即不能分配资源（0，2，0）给 $P_1$。