**1.**

可重用资源：

一次仅供一个进程安全使用且不因使用而耗尽的资源。如处理器、I/O通道、内存和外存、设备，以及诸如文件、数据库和信号量之类的数据结构之类的

可消耗资源：

可消耗资源是指可被创建（生产）和销毁（消耗）的资源。如中断、信号、消息和I/O缓冲中信息

**2.**

死锁的充分必要条件：互斥访问，非抢占，占有并等待，循环等待

**3.**

要求进程一次性请求所有资源，并阻塞这个进程直到所有资源请求能够满足

但是存在效率较低的问题，进程可能会阻塞很长时间，分配给进程的资源很长时间内不会被使用。而且大部分情况下事先不知进程所需的全部资源

**4.**

定义一个请求资源的顺序；系统把所有资源按类型进行线性排队，如Ri, Rj, Rk (i<j<k)；所有进程对资源的请求必须严格按资源序号递增的顺序提出，效率较低。

**5.死锁避免、检测和预防之间的区别是什么**

三者发生的时机不同，死锁预防发生在运行前，操作系统对资源的分配策略做出了一定的限制，破坏了死锁的必要条件。死锁避免发生在进程运行前，操作系统对进程申请的资源进行检测验证，保证系统不会进入到不安全状态。死锁检测发生在进程运行时，按照某种策略处理死锁，比较灵活。

**6.**

安全状态是指至少有一个资源分配序列（Px,Py,…,Pz，安全序列）不会导致死锁，按安全序列为进程分配资源，所有进程Px,Py,…,Pz能够运行结束

不安全状态: 不存在安全序列

**7.**

银行家算法设计思想：

当用户申请一组资源时，系统必须做出判断：如果把这些资源分出去，系统是否还处于安全状态。若是，就可以分配这些资源；否则，暂时不分配，阻塞进程。

银行家算法详述：

设 Request；是进程Pi的请求向量，如果 Requesti[j] = K，表示进程Pi需要K个Rj类型的资源。当Pi发出资源请求后，系统按下述步骤进行检査:

(1) 如果 Requesti[j] ≤ Need[i,j]便转向步骤(2)；否则认为出错，因为它所需要的资源数已超过它所宣布的最大值。

(2) 如果 Requesti[j] ≤ Available[j]，便转向步骤(3)；否则，表示尚无足够资源，Pi须等待。

(3) 系统试探着把资源分配给进程Pi，并修改下面数据结构中的数值

Available[j] = Available[j] - Requesti[j];

Allocation[i,j] = Allocation[i,j] + Requesti[j];

Need[i,j] = Need[i,j] - Requesti[j];

(4) 系统执行安全性算法，检查此次资源分配后系统是否处于安全状态。若安全，才正式将资源分配给进程Pi，以完成本次分配；否则，将本次的试探分配作废，恢复原来的资源分配状态，让进程Pi等待。

**8.请简述死锁检测的流程**

死锁检测算法步骤

1.标记Allocation矩阵中一行全为零的进程；

2. 初始化一个临时向量W，令W等于Available向量；

3. 查找下标i, 进程i当前未标记且满足Q(Qij表示进程i请求j类资源的数量）的第i行小于等于W, 即对所有的1<=k<=m, Qik<=Wk, 若找不到这样的行，终止算法；

4. 若找到这样的行，标记进程i，并把Allcation矩阵中的相应行加到W中，即对所有1<=k<=m, 令Wk= Wk+Aik。返回3

当且仅当最终有未标记进程时，才存在死锁，未标记的进程都是死锁相关进程

**9.**(1)T0为安全状态，安全序列如下：（安全序列不唯一，只要答对一个即可）P4->P2->P3->P5->P1

(2)请求Request2=(0,3,4),还需资源need2=(1,3,4),可用资源available=(2,3,3)

满足Request2≤need2,但不满足Request2≤available,故不能分配资源

(3) Request4=(2,0,1)，need4=(2,0,1)，可用资源available=(2,3,3)

满足Request4≤need4

满足Request4≤available

试探性分配后，执行安全检测算法，系统存在安全状态，因此可以分配资源

解答过程示例：

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **最大资源需求** | | | **已分配资源数量allocation** | | | **还需 need** | | | **available** | | |
| **A** | **B** | **C** | **A** | **B** | **C** | **A** | **B** | **C** | **A** | **B** | **C** |
| **P1**  **P2**  **P3**  **P4**  **P5** | **5**  **5**  **4**  **4**  **4** | **5**  **3**  **0**  **0**  **2** | **9**  **6**  **11**  **5**  **4** | **2**  **4**  **4**  **2**  **3** | **1**  **0**  **0**  **0**  **1** | **2**  **2**  **5**  **4**  **4** | **3**  **1**  **0**  **2**  **1** | **4**  **3**  **0**  **0**  **1** | **7**  **4**  **6**  **1**  **0** | **2** | **3** | **3** |

安全性算法检测（安全序列不唯一，这里仅仅是一个安全序列示例）

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Work** | | | **need** | | | **allocation** | | | **Work+allocation** | | | **finish** |
| **A** | **B** | **C** | **A** | **B** | **C** | **A** | **B** | **C** | **A** | **B** | **C** |  |
| **P4**  **P2**  **P3**  **P5**  **P1** | **2**  **4**  **8**  **12**  **15** | **3**  **3**  **3**  **3**  **4** | **3**  **7**  **9**  **14**  **18** | **2**  **1**  **0**  **1**  **3** | **0**  **3**  **0**  **1**  **4** | **1**  **4**  **6**  **0**  **7** | **2**  **4**  **4**  **3**  **2** | **0**  **0**  **0**  **1**  **1** | **4**  **2**  **5**  **4**  **2** | **4**  **8**  **12**  **15**  **17** | **3**  **3**  **3**  **4**  **5** | **7**  **9**  **14**  **18**  **20** | **True**  **True**  **True**  **True**  **True** |