**学号：2016011984 姓名：武傲**

**操作系统课第一次课后作业 2018.4**

1. 找出死锁和饥饿的定义和区别，并指明出处；自己做出通俗解释（举大白话例子）。
2. 死锁（Deadlock）：死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。(出自百度百科 ‘死锁’ 词条，该词条由[“科普中国”百科科学词条编写与应用工作项目](http://www.cast.org.cn/) 审核)
3. 饥饿（Starvation）: 指系统不能保证某个进程的等待时间上界，从而使该进程长时间等待，当等待时间给进程推进和响应带来明显影响时，称为发生了进程饥饿。当饥饿到一定程度的进程所赋予的任务即使完成也不再具有实际意义时称该进程被饿死。（出自百度知道）
4. 相同点：二者都是由于竞争资源而引起的。

不同点：

* 1. 从[进程状态](https://www.baidu.com/s?wd=%E8%BF%9B%E7%A8%8B%E7%8A%B6%E6%80%81&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLn1P-uh79m1IhnWfsuHwb0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWDsPj6dPjmY" \t "_blank)考虑，死锁进程都处于等待状态，忙等待(处于运行或就绪状态)的进程并非处于等待状态，但却可能被饿死；
  2. 死锁进程等待永远不会被释放的资源，饿死进程等待会被释放但却不会分配给自己的资源，表现为等待时限没有上界(排队等待或忙式等待)；
  3. 死锁一定发生了循环等待，而饿死则不然。这也表明通过资源分配图可以检测死锁存在与否，但却不能检测是否有进程饿死；
  4. 死锁一定涉及多个进程，而饥饿或被饿死的进程可能只有一个。
  5. 在饥饿的情形下，系统中有至少一个进程能正常运行，只是饥饿进程得不到执行机会。而死锁则可能会最终使整个系统陷入死锁并崩溃。（出自百度知道）

1. 通俗解释：死锁举个例子：一群人去上厕所，坑位不够，占到位置的人没有带纸，而带纸的人都没有位置，（前提人们没有沟通或让步），接下来陷入僵局，这就是死锁，他们陷入循环式的等待。饥饿是某个进程由于当前资源分配先后方式一直排不到它，比如来一个比它先，再来一个还比它先，一直这样，过了很久这个进程已经没意义了就是饿死了。死锁指一群进程因僵持不下而一起等，而饥饿指某进程一直等还排不到它。



（1）T0时刻的安全性：利用安全性算法对T0时刻的资源分配情况进行分析，如下图所示。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 资源  进程 | Work | Need | Allocation | Work+Allocation | Finish |
| A B C | A B C | A B C | A B C |
| P4 | 2 3 3 | 2 2 1 | 2 0 4 | 4 3 7 | true |
| P2 | 4 3 7 | 1 3 4 | 4 0 2 | 8 3 9 | true |
| P3 | 8 3 9 | 0 0 6 | 4 0 5 | 12 3 14 | true |
| P5 | 12 3 14 | 1 1 0 | 3 1 4 | 15 4 18 | true |
| P1 | 15 4 18 | 3 4 7 | 2 1 2 | 17 5 20 | true |

所以T0时刻存在一个安全序列 {P4，P2，P3，P5，P1}，故系统是安全的。

（2）T0时刻，① P2：Request（0，3，4）≤Need2（1，3，4）；

② P2：Request（0，3，4）> Available（2，3，3），所以让P2等待。

（3）在（2）的基础上，

① P4：Request（2，0，1）≤Need4（2，2，1）；

② P4：Request（2，0，1）≤Available（2，3，3）；

③系统先假定可为P4分配资源，并修改有关数据为： Allocation4（4，0，5）， Need4(0，2，0)，Available（0，3，2），进行安全性算法检查，如下图所示。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 资源  进程 | Work | Need | Allocation | Work+Allocation | Finish |
| A B C | A B C | A B C | A B C |
| P4 | 0 3 2 | 0 2 0 | 4 0 5 | 4 3 7 | true |
| P2 | 4 3 7 | 1 3 4 | 4 0 2 | 8 3 9 | true |
| P3 | 8 3 9 | 0 0 6 | 4 0 5 | 12 3 14 | true |
| P5 | 12 3 14 | 1 1 0 | 3 1 4 | 15 4 18 | true |
| P1 | 15 4 18 | 3 4 7 | 2 1 2 | 17 5 20 | true |

由上图可知，可以找到一个安全序列 {P4，P2，P3，P5，P1}，故系统是安全的，可以立即将P4所申请的资源分配给它。

（4）在（3）的基础上为Allocation4（4，0，5）， Need4(0，2，0)，Available（0，3，2）；

① P1：Request（0，2，0）≤Need1（3，4，7）；

② P1：Request（0，2，0）≤Available（0，3，2）；

③ 系统先假定可为P1分配资源，并修改有关数据为： Allocation1（2，3，2）， Need1(3，2，7)，Available（0，1，2），进行安全性算法检查，此时系统状态如下图所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 资源  进程 | Allocation | Need | Available |
| A B C | A B C | A B C |
| P1 | 2 3 2 | 3 2 7 | 0 1 2 |
| P2 | 4 0 2 | 1 3 4 |  |
| P3 | 4 0 5 | 0 0 6 |  |
| P4 | 4 0 5 | 0 2 0 |  |
| P5 | 3 1 4 | 1 1 0 |  |

由上图可知，可用资源Available（0，1，2）已不能满足任何进程的需要，故系统进入不安全状态，此时系统不分配资源给P1。