

哈工大操作系统-L19死锁处理

哈工大操作系统-L19死锁处理

1. 什么叫死锁
2. 死锁的成因
3. 死锁的4个必要条件
4. 死锁的4种处理方法
 - 4.1 死锁预防
 - 4.2 死锁避免(银行家算法)
 - 4.3 死锁检测+恢复
 - 4.4 死锁忽略

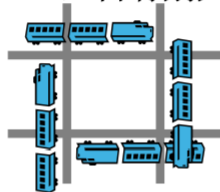
1. 什么叫死锁

死锁：多个进程由于互相等待对方持有的资源，而造成的谁都无法执行的情况。

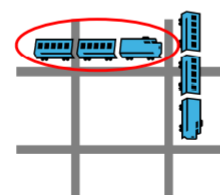
2. 死锁的成因

死锁的成因

- 资源互斥使用，一旦占有别人无法使用

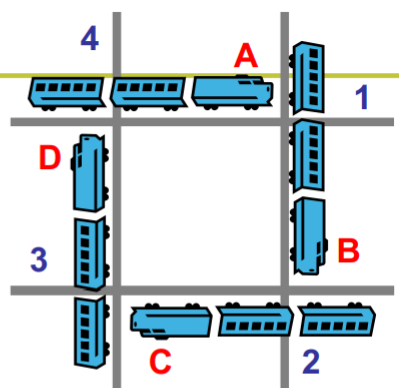


- 进程占有了一些资源，又不释放，再去申请其他资源



- 各自占有的资源和互相申请的资源形成了环路等待

- 资源互斥使用
- 多个进程互相等待对方资源造成环路等待



3. 死锁的4个必要条件

死锁的4个必要条件

■ 互斥使用(Mutual exclusion)

- 资源的固有特性，如道口

■ 不可抢占(No preemption)

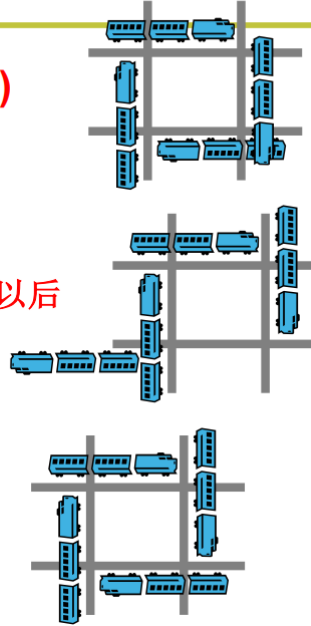
- 资源只能自愿放弃，如车开走以后

■ 请求和保持(Hold and wait)

- 进程必须占有资源，再去申请

■ 循环等待(Circular wait)

- 在资源分配图中存在一个环路



- 互斥使用
- 不可抢占
- 请求和保持--必须先占有资源再去申请
- 循环等待

4.死锁的4种处理方法

死锁处理方法概述

■ 死锁预防 “no smoking”，预防火灾

- 破坏死锁出现的条件

■ 死锁避免 检测到煤气超标时，自动切断电源

- 检测每个资源请求，如果造成死锁就拒绝

■ 死锁检测+恢复 发现火灾时，立刻拿起灭火器

- 检测到死锁出现时，让一些进程回滚，让出资源

■ 死锁忽略 在太阳上可以对火灾全然不顾

- 就好像没有出现死锁一样

- 死锁预防--破坏死锁的四个必要条件
- 死锁避免--不破坏必要条件，在申请资源时审查是否申请的资源会造成死锁
- 死锁检查+恢复--死锁产生后进行恢复
- 死锁忽略

4.1死锁预防

死锁预防的方法例子

- 在进程执行前，**一次性申请所有需要的资源**，不会占有资源再去申请其它资源

- 缺点1: 需要预知未来，编程困难
- 缺点2: 许多资源分配后很长时间后才使用，资源利用率低

- 对资源类型进行排序，**资源申请必须按序进行**，不会出现环路等待

- 缺点: 仍然造成资源浪费

问题: 为什么使用这两种方法，一定不会死锁?

4.2死锁避免(银行家算法)

死锁避免: 判断此次请求是否引起死锁?

- 如果系统中的所有进程存在一个可完成的执行序列 P_1, \dots, P_n ，则称系统处于**安全状态**

都能执行完成当然就不死锁

- **安全序列**: 上面的执行序列 P_1, \dots, P_n **如何找?**

已经申请到的资源数目

当前可用的资源数

	Allocation	Need	Available
	A B C	A B C	A B C
P0	0 1 0	7 4 3	2 3 0
P1	3 0 2	0 2 0	
P2	3 0 2	6 0 0	
P3	2 1 1	0 1 0	
P4	0 0 2	4 3 1	

问题: 下面哪个是安全序列()?

- A. P1, P3, P2, P4, P0
- B. P0, P1, P2, P3, P4
- C. P3, P0, P1, P2, P4
- D. P3, P4, P1, P2, P0

找安全序列的银行家算法(Dijkstra提出)

```
int Available[1..m]; //每种资源剩余数量
int Allocation[1..n,1..m]; //已分配资源数量
int Need[1..n,1..m]; //进程还需的各种资源数量
int Work[1..m]; //工作向量
bool Finish [1..n]; //进程是否结束
```

```
Work = Available; Finish[1..n] = false;
while(true){
    for(i=1; i<=n; i++){
        if(Finish[i]==false && Need[i]≤Work){
            Work = Work + Allocation[i]; //资源个数
            Finish[i] = true; break;}
        else {goto end;}
    }
}
End: for(i=1;i<=n;i++)
    if(Finish[i]==false) return "deadlock";
```

$T(n)=O(mn^2)$

资源个数
进程个数

请求出现时: 首先假装分配, 然后调用银行家算法

■ P_0 申请(0,2,0) 假装分配

	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 3 0	7 2 3	2 1 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- 进程 P_0, P_1, P_2, P_3, P_4 一个也没法执行, 死锁进程组
- 此次申请被拒绝

4.3死锁检测+恢复

死锁检测+恢复: 发现问题再处理

- 基本原因: 每次申请都执行 $O(mn^2)$, 效率低。发现问题再处理
 - 定时检测或者是发现资源利用率低时检测

```
Finish[1..n] = false;  
if(Allocation[i] == 0) Finish[i]=true;  
...//和Banker算法完全一样  
for(i=1;i<=n;i++)  
    if(Finish[i]==false)  
        deadlock = deadlock + {i};
```

- 选择哪些进程回滚? 优先级? 占用资源多的? ...
- 如何实现回滚? 那些已经修改的文件怎么办?

4.4死锁忽略

死锁忽略的引出

- 死锁预防?
 - 引入太多不合理因素...
- 死锁避免?
 - 每次申请都执行银行家算法 $O(mn^2)$, 效率太低
- 死锁检测+恢复?
 - 恢复很不容易, 进程造成的改变很难恢复
- 死锁忽略
 - 死锁出现不是确定的, 又可以用重启来处理死锁
 - 有趣的是大多数非专门的操作系统都用它, 如 UNIX, Linux, Windows