进程的并发控制和死锁

并发控制的特点

- 资源共享 - 互斥关系

  间接制约关系

- 协作 - 同步关系

  直接制约关系

- 前序关系

  直接和间接关系决定了进程的顺序时间关系

  - 顺序关系 : S(P1,P2,..)
  - 并行关系 : P(P1,P2,P3,P4,...)
  - 一般关系(顺序关系 + 并行关系) : S(P(P1,P2,...),Pn)

进程通信

进程通信 :

- 进程之间交换信息
- 进程之间的相互制约和以来和合作的关系，我们又称之为是进程同步，互斥是同步的一种情况，又称进程的低级通信

进程互斥

1. 临界区 : 并发进程访问临界资源的那段必须互斥执行的程序
2. 临界资源 : 一次仅允许一个进程使用的系统中的共享资源
3. 程序进入临界区的准则
   1. 互斥使用 : 不存在两个以上包括两个程序都在临界区执行
   2. 让权等待 : 等待进入临界区应当放弃处理机阻塞
   3. 有空让进 : 临界区外的程序进入临界区不受限制
   4. 有限等待 : 进程在临界区外等待是有限时间的，不可无限等待

互斥的硬件实现

- 关中断 : 进程在临界区执行的时候，关闭所有的中断
  - 实现简单
  - 代价过高，限制了程序的交叉处理能力(并行性下降)
- Test&Set : 测试和设置指令是一条不可中断的机器指令

  设置锁位变量w(0资源可用，1资源不可用)
  - 实现简单有效
  - 进程循环测试锁位，CPU的时间被浪费，不是阻塞等待(忙等待)

互斥的软件实现

1. 共享变量
   1. $flag[2]$
   2. $turn$
2. 线程 :

   P_i, P_j

3. 初始化

$flag[1] = flag[0] = false, turn = i/j$

4. 描述
    - P_i如果进入临界区，$flag[i] = true, turn = j$,表示P_i准备好了礼让P_j进入临界区
    - $flag$若均为$true$根据$turn$决定那个进程进入临界区
5. 好处
    - 不需要系统调用的开销
    - 该方法使用在线程控制中，保持共享变量的一致性

信号量和**PV**操作

- 基本原理 : 两个或者多个进程通过简单的信号的通信信号合作，一个进程被迫在某一处停止直到接收到特定的信号
- 信号量表示系统共享资源的物理实体

```
typedef struct{
  int   value;       // 共享资源的数量
  struct    process* list;       // 排队的队头指针
}
```

- 只允许的操作
    - P :
        - value--
        - value >= 0 : 进程执行对应的操作，不阻塞，否则加入阻塞队列
        - 申请资源,等待
    - V :
        - value++
        - value <= 0 : 释放一个阻塞进程然后转入处理机调度，否则继续进程的执行
        - 释放资源，发送信号
- 信号量实现进程互斥
    - 为了解决并发进程的互斥共享问题，我们引入一个互斥信号量，初值是1
    - 等待的进程的数目就是信号量的绝对值

```
mutex:integer:=1
  parbegin
  p1:begin
    P(mutex)
    (critical sction)
    V(mutex)
    ...
  end
  p2:begin
    P(mutex)
    (critical section)
    V(mutex)
    ...
  end
parend
```

- 信号量的同步操作
    - P,V操作分散在多个进程之间执行
    - 使用V操作来发送信号

经典问题(核心在于找到对应的消费者和生产者和产品)

- 生产者消费者问题
    1. M个生产者和N个消费者通过一个有界环形缓冲区发生联系，缓冲区的大小是K
    2. 生产者和消费者之间互斥使用
    3. 问题解决
- 读者写者问题

进程高级通信

优缺点

- 低级通信速度快传送信息量小并且效率底下
- 低级通信使用不当的话容易导致死锁的发生
- 容易忘掉释放资源(V)

通信方式

- 消息缓冲
    - 发送原语
    - 接收原语
- 信箱

  一个信箱可以容纳你若干邮件
    - 发送信件 : 制定的信箱没有满可以发送，否则发送失败阻塞等待
    - 接收信件 : 如果有信取走并检查是否有发送者在等待，有则唤醒
- 管道 : 临时文件
    - 连接写进程和读进程的共享文件
    - 先进先出的方式实现通信，OS内部的核心缓冲区实现
    - 系统调用,返回2个文件描述符
    - 临时文件存在于内存中而不是磁盘上
    - 形式
        - 命令方式的管道
        - 程序方式的管道
- 共享内存
    - 最快捷的方式
    - 交换大量数据，在主存中画出一块共享内存区，并将该共享内存区连接到各自地址空间的某一部分

发送方式

- 非阻塞发送，阻塞接收 : 单向通信
- 非阻塞发送，非阻塞接收 : 广泛使用，单向通信
- 阻塞发送，阻塞接收 : 双向通信

死锁

- 资源的特性
    - 可抢占式资源
        - 资源剥夺后，对进程不会产生什么影响(保存现场)
        - 主存 / CPU
    - 不可抢占式资源
        - 临界资源 : 剥夺会导致程序的运行失败
    - 死锁一般涉及的都是不可抢占式资源
- 死锁的定义
    - 一组中的每一个进程都在等待这一组中的其他进程锁占有的资源可能引起的错误现象

- 死锁产生的必要条件
    - 互斥条件：资源不可共享
    - 保持和等待条件：进程因为申请资源被阻塞对其原本拥有的资源保持不放
    - 不剥夺条件：进程不可以被剥夺资源，除非被自己释放
    - 循环等待条件：存在进程循环链，每一个进程都在等待下一个进程保持的资源
- 死锁的原因

  产生死锁的根本原因：是对独占资源的共享，并发进程的同步关系不当。

    - 系统资源配置不足
    - 系统的并发请求资源的随机性
        - 资源类别
        - 资源数量
    - 进程异步执行，造成进程推进顺序的非法性
- 解决死锁的算法
    - 鸵鸟算法
    - 死锁预防

      破坏死锁的必要条件

        - 破坏互斥条件：改造设备变成共享设备，但是仍有可能导致死锁
        - 破坏保持等待条件：进程运行前获得所有的资源，利用率低不合理
        - 破坏不剥夺条件：放弃资源的代价很高
        - 破坏循环等待条件：打乱环即可，不可能有效
    - 死锁的避免
        - 动态的申请资源，一次申请一部分，计算安全性
            - 安全分配（存在一个进程完成序列）
            - 不安全阻塞
        - 进程资源轨迹图
        - 银行家算法：进程完成序列找到才算数
    - 死锁的检测和恢复
        - 不阻止死锁但是检测和恢复死锁
        - 死锁的检测

          对系统中的进程的资源构成的图检查，存在环路则环路中的任何一个进程都是死锁的，没有环路则无死锁
        - 死锁的恢复
            - 故障终止一些进程
                - 终止所有：开销小，损失大
                - 终止一个：开销大，损失小

              最好故障终止一些影响不大的进程
                - 刚启动的进程
                - 输出最少进程
                - 撤销的时候系统耗费的处理机时间最少
                - 分配资源总量最少
                - 进程的优先级低
            - 资源剥夺

              剥夺一部分的资源并交给其他的进程，直到死锁的恢复，然后将断点恢复是被剥夺的进程重新运行
                - 取走一些资源
                - 回滚一些进程：将一死锁进程滚回到获得资源之前的执行点,为进程设置执行点是指将进程在该点的执行状态信息写到一个文件中，便于以后从该执行点启动进程执行。