进程同步、互斥、死锁

死锁

发生条件(需同时满足)

- 互斥：对必须互斥使用的资源进行争抢
- 非抢断：一个进程在获得的资源使用完前，不能被其他进程抢占，除非自己放弃
- 请求和保持(占用并等待)：进程已经保持至少一个资源，但又提出需要新资源的请求，而该资源当前被占用；此时请求进程被阻塞，但已拥有的资源保持，不放回
- 等待循环：资源的循环等待链，链种一个进程拥有的资源被下一个进程请求(有环)

解决

不允许死锁

静态策略

预防死锁：破坏4个发生条件中的几个

- 破坏互斥条件
  - 如使用SPOOLing技术，使独占设备成为共享设备
  - 缺点：只有少数设备可以这么做，如打印机
- 破坏非抢断
  - 当一个进程申请的资源被另一个进程占有，可以按照一定规则抢占
  - 当一个进程申请的资源被另一个进程占有，主动释放自己当前拥有的资源，等到以后有机会再重新调度进程
  - 缺点
    - 实现复杂
    - 主动释放资源可能导致进程前功尽弃，因此该方式仅适用易保存和恢复状态的资源，如CPU
    - 反复申请释放资源会增大开销，降低吞吐量
    - 反复主动释放可能导致饥饿
- 破坏请求和保持
  - 静态分配：进程运行前一口气全部申请全部资源
  - 缺点：有的资源使用时间很小，但是一直被进程占用，资源利用率低。且可能导致饥饿
- 破坏循环等待
  - 给系统资源编号，进程按照编号顺序申请资源
  - 缺点
    - 实际资源使用顺序可能不同，导致资源浪费
    - 不方便添加新设备，因为要重新编号

动态策略

避免死锁：预防系统进入不安全状态(银行家算法)

- 安全序列：安全分配资源的序列
- 在分配资源前，判断是否遵循安全序列
- 银行家算法

允许死锁

检测与解除：如果发现死锁则解决

- 资源剥夺：将一个死锁进程挂起，将它的资源分给其他死锁进程
- 撤销进程：直接撤销死锁进程，实现简单，但是会使进程死锁前的工作全部白费
- 进程回退：按照系统记录进程的执行历史信息，回退到死锁前的时候

检测的乐观假设

- 在资源分配方面和进程请求资源的过程中，不存在任何形式的循环等待。
- 如果在实际过程中，一个循环等待确实发生，这种假设可以被违反。

信号量机制

弱化：互斥锁

- bool值
- 忙等待
- 自旋锁

定义

通过OS提供的一对原语【wait(S)/P(S)、signal(S)/V(S)】处理信号量

- wait相当于进入区
- signal相当于退出区
- 信号量≤0要释放一个的等待进程

信号量定义

可以表示系统某种资源数量

信号量类型

整型

只能被初始化、P、V改变的整数型变量　　加一减一

- 表示系统某种资源数量
- 检查上锁一气呵成，防止异步、导致问题并发
- 但会忙等待，不满足让权等待

记录型

遵循让权等待，不会忙等待

```
记型信号量的缺陷是存在“忙等”问题，因此人们又提出了“记录型信号量”，即用记录型数据结构表
示的信号量。

*记录型信号量的定义*/
ypedef struct {
 int value;          //剩余资源数
 struct process *L;  //等待队列
 semaphore;

*某进程需要使用资源时，通过 wait 原语申请*/
oid wait (semaphore S) {
 S.value--;
 if (S.value < 0 ) {
   block (S.L);
 }
```

如果剩余资源数不够，使用block原语使进程从运行态进入阻塞态，并把该进程插入等待该资源的等待队列（即阻塞队列）中

```
/*进程使用完资源后，通过 signal 原语释放*/
void signal (semaphore S) {
  S.value++;
  if (S.value <= 0) {
    wakeup(S.L);
  }
}
```

释放资源后，若还有别的进程在等待这种资源，则使用 wakeup 原语唤醒等待队列中的一个进程，该进程从阻塞态变为就绪态

临界区问题

临界资源：一个时间段只允许一个进程访问(如打印机、摄像头)

定义

- 每个进程都有处理共享数据或资源的代码段，这称为临界区
  - 当这部分代码段被执行时，称进程处于该共享数据/资源的临界区
- 间接制约关系，对临界资源的访问必须互斥(一个访问另一个等待)

原则

- 空闲让进：临界区空闲，可让一个请求立即进入
  - 又叫做：推进
- 忙则等待：已有进程在临界区，其他要等
  - 可理解为：互斥
- 有限等待：对与等待中的进程，保证在有限时间内可以进入临界区(防止饥饿)
  - 处理：如轮流访问
- 让权等待：进程不能进入临界区，则立即释放处理机，防止进程忙等待

临界区代码结构

必须原子操作实现

- 进入区：检查是否可以进入临界区(是否可以访问该临界资源)，设置正在访问标志(上锁)
- 临界区：访问临界资源
- 退出区：解除正在访问的标志(解锁)
- 剩余区：其他处理

处理

抢占式内核

- 允许被抢占
- 响应速度更快
- 更适合实时编程

非抢占式内核

- 内核模式下进程不允许抢占
- 不受内核数据结构影响

实现

软件

问题都因为进程的并发、异步

单标志

- 一个进程访问完，将临界区的权限给另一个进程(每个进程访问临界区的权限只能另一个进程给)
- 违背空闲让进

双标志先检查

- 设置一个数组存进程进入临界区的“意愿”，全先设“无意愿”，每个进程进入临界区前，先找其他有意愿的进程将其进入临界区，如果没有则设置自己“有意愿”
- 违背忙则等待
  - 检查和上锁过程不是一气呵成，可能在不同进程的检查上锁间发生切换

双标志后检查

- 先上锁后检查，每个进程进入临界区前先上锁，再找其他进程意愿
- 违背空闲让进、有限等待

Peterson算法

- 除了设置“意愿”数组，多设置让谁先访问的标识，在有意愿进入临界区时同时表示可以让其他进程先访问。(双标志后检查+孔融让梨)
- 为遵循让权等待

Lamport烘焙算法

- 每个进程会接收编号，先按编号小的，再按pid排序

硬件

中断屏蔽

- 关中断(当前进程不会被中断/切换)->临界区->开中断
- 优点：简单、高效
- 缺点：不适合多处理机/只适合OS内核进程，不适合用户进程
  - 因为只禁止了一个cpu的中断，其他没关，所以临界区是原子的但不是互斥的

TestAndSet/TestAndSetLock(TS指令/TSL指令)

- 设置bool型变量lock存在寄存器，设置临界区上锁
- 优点：实现简单，适用多处理机
- 缺点：不满足让权等待，等待中的进程会占用CPU并循环执行TSL指令，导致忙等待（饥饿）

Swap指令(XCHG[Excahnge]指令)

- 设置old存以前的lock值，用于判断当前临界区是否上锁，逻辑上同TSL，仅是硬件实现不同
- 优缺点同TSL

操作系统

编程语言

相关临界区

多个进程(并发进程)中涉及到同一个临界资源的临界区

管程monitor

信号量机制容易出错且难编写

抽象数据类型(ADT)

- 局部于管程的共享数据结构说明
- 对共享数据结构进行操作的一系列函数
- 对局部于管程的共享数据设置初始值的语句
- 管程名

特征

- 与“class类”类似
- 局部于管程的成员只能由局部与管程的过程访问
  - private成员只能由class内部函数改变
- 一个进程只能通过调用管程内的函数才能访问管程共享数据
  - 要声明一个class在进行后续操作
- 每次(同一时间)仅允许一个进程在管程内执行某个管程成员函数
  - 由编译器实现相关互斥

竞争条件

- 发生：多个进程并发(同时)访问操作共享数据
- 为防止竞争条件需要同步进程

为了防止死锁和饥饿

- 临界区进程不可无限期延迟
- 解决方式：设置临界区
- 死锁：所有进程都在等待另一个进程
- 饥饿：两个进程一直相互发送消息，而其他进程只能一直等待

同步

直接制约关系，一个进程的执行以来另一个进程的消息，并发进程间必然的依赖，协调不同进程因工作次序产生的制约关系(解决进程异步性的不可预知)

访问临界资源基本关系

使用信号量机制实现

- 为了使本来异步进行的进程相互配合，有序推进
  - 保证两个操作“一前一后”执行
    - 在“前一个操作”之后执行V(S)
    - 在“后一个操作”之前执行P(S)
  - 设置同步信号量S，处值为0
- 将其他进程需要的消息发出，也能测试进程本身需要的消息是否收到
  - wait/signal
    - wait测试消息是否到达
    - signal发消息出去
  - 进程同步一个消息对应一个信号量
- 进程在不同信号量V操作可把不同消息发送出去

信号量实现前驱关系

- 操作按照前驱图顺序执行
- 每一对前驱关系就是一个同步问题(一前一后)
- 每一对前驱关系的PV与同步实现一样
  - 为每一对前驱关系设置一个同步变量

进程同步的特殊情况

使用信号量机制实现

- 设置临界区和信号量mutex，初值为1
  - Semaphore mutex=1
  - 在临界区之前执行P(mutex)
  - 在临界区之后执行V(mutex)
  - 不同临界资源要有不同信号量
- 用一个信号量与一组相关临界区

互斥

经典问题

有限缓冲区

生产者消费者

系统有一组生产者一组消费者

同步关系

- 缓冲区没满才能放进缓冲区，否则等待
  - 生产者每次生产一个数据进入缓冲区
- 缓冲区不空才能取，否则等待
  - 消费者每次从缓冲区取一个

互斥关系

- 缓冲区是临界资源，要互斥访问
  - 生产者消费者共享缓冲区

使用信号量机制处理

即互斥要在同步之后

- 先p(mutex)会发生阻塞
- 生产者p(empty)v(full)
- 消费者p(full)v(empty)

先P(empty)/P(full)再P(mutex)

- 一个互斥信号量，值为1(用于缓冲区互斥访问)
- 两个同步信号量，值为0(full数据数量)和n(empty空闲缓冲区的数量)

采用多线程技术将消费者变为进程中的两个线程

- 两个线程有并发性
- 不需要额外公共缓冲区

多生产者多消费者

多类生产者消费者

- 生产者生产不同类数据，而消费者也只消费对应类型数据
- 缓冲区初始化为1时，可以不用mutex(互斥信号量)

吸烟者问题

一个生产者对多个消费者

- 生产者会轮流为消费者分别提供各自所需的数据
- 直到当前消费者取走并完成所需数据和操作，生产者才会为下一个消费者提供数据

读者作者问题

多个读者和作者共享一块内存(只有读者可以改)

- 多个读者访问共享数据不需要互斥，因为不会改变内容
- 但是作者必须与其他作者和或读者互斥
- 设置一个互斥信号量rw为写者，一个mutex互斥为读者，一个记录当前访问读者数count

哲学家用餐

一个哲学家需要两个临界资源，分配不当会导致死锁

- 设置互斥信号量数组

解决死锁

- 添加如:只允许4个哲学家同时拿筷子等限制
- 奇数号哲学家只能先拿左边筷子再右边，偶数号先右后左
- 仅当哲学家左右筷子都可用才能拿起筷子