操作系统同步问题

▼ 进程的同步,互斥,临界区

同步

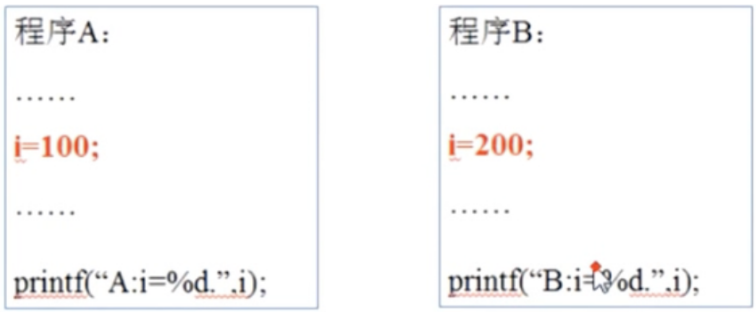
若于合作进程为了完成一个共同的任务，需要相互协调运行进度，一个进程开始某个操作之前，必须要求另一个进程已经完成某个操作，否则前面的进程只能等待。
例如，学生课程学习考试包括四个功能：教师出卷、教师改卷：学生考试、学生订正。
则正确流程为：教师出卷一学生考试一教师改卷一学生订正。
同步关系要求：
学生考试前，必须完成教师出卷，否则等待；
教师改卷前，必须完成学生考试，否则等待：
学生订正前，必须完成教师改卷，否则等待

互斥

多个进程共享了独占性资源,必须协调各进程对资源的存取顺序,确保没有任何两个或者以上的进程同时进行存取操作

i是全局变量

资源称为临界资源,存取操作区域称为临界区



临界资源

一次只允许一个进程使用的共享资源称为临界资源

许多硬件资源如打印机,磁带机都是临界资源

进程之间需要采用互斥方式实现对这些资源的共享

临界区

在每个进程中访问临界资源的代码为临界区

```
while(TRUE){
  进入区
  临界区
  退出区
  剩余区
}
```

临界区进入准则
为实现进程互斥地进入自己的临界区，可用软件方法，更多的是在系统中设置专门的同步机构来协调各进程间的运行。所有同步机制都应遵循下述四条准则：
1)单个入区。一次仅允许一个进程进入

2)独自占用。处于临界区内的进程不可多于一个。如果已有一个进入临界区，其它试图进入的进程必须等待。

3） 尽快退出。访问完后尽快退出，以让出资源。
4） 落败让权。如果进程不能进入临界区，则应让出CPU，以免出现“忙等” 现象。

▼ 互斥的实现方式

▼ 锁机制

设置一个标志锁w,表明临界区是否可用0 1

上锁: 进入临界区前,检查标志锁w是否可用

若 不可用 进程在临界区外等待

若 可用

将标志锁w设为不可用

访问临界资源

开锁: 推出临界区前,将标志锁w设置为可用

```
上锁
1,检查锁状态,如果w为1则返回此步循环
2,如果w=0,则设置为1
```

```
上锁lock(W)
  while(W==1);
  W=1;

开锁unlock(W)
  w = 0
```

用锁访问临界区举例

```
程序A : int W = 0

```
lock(W); //第二句赋值为1
i = 100;
```
printf(A:i=%d"i)
unlock(W)
```

程序B:
```
lock(W);
i = 100;
```
printf(A:i=%d"i)
unlock(W)
```
```

忙则等待.空闲让进,有限等待满足,让权等待不满足,如果等不到,就一直while循环

▼ 硬件同步机制

1,禁止中断

方法：关中断是实现互斥的最简单的方法之一。每个进程在进入临界区之后关闭所有中断，在离开临界区之前才重新打开中断。

原理：由于禁止中断，时钟中断也被禁止。这样就不会把CPU切换到另外的进程。不必担心其它进程对它的干扰。

缺点：①滥用关中断权力可能导致严重后果：②关中断时间过长，会影响系统效率，限制了处理器交叉执行程序的能力：③关中断方法也不适用于多CPU系

统，因为在一个CPU上关中断并不能防止进程在其它CPU上执行相同的临界段代码。

2, test and set指令实现

指令的执行过程不可分割

```
boolean TSL(boolean *lock)
{
boolean old;
old = *lock
*lock = ture
return old;
}
```

▼ 原语PV

原子操作,要么全做,要么不做

原语操作不允许并发,代码短

pv是原子操作,执行时不可中断

对信号量S的操作限制

信号量S可以初始化为一个非负值

只能由PV两个操作来访问信号量

```
p操作
P(S){
  while(S<= 0)
  S--
}

v操作
V(S){
  S++
}
```

▼ 信号量机制

整型信号量 结构型信号量 信号量集

1,整型信号量

P proberen 测试 wait(S) 探测

V verhogen 增加 signal(S) 释放

s可以初始化为非负值

只能由pv两个操作来访问信号量

```
p操作
P(S){
  while(S<= 0) //while就是去占用,比如s有5,可以进入5个进程,代表能占用资源的个数
  S--
}

v操作
V(S){
  S++
}

p在临界区前,v在临界区后
```

2,记录型信号量

```
struct{
  int value
  struct process list
}semaphore

p(semaphore *S){
  S->value--
  if(S->value<0)block(S->list)
}

v(semaphore *S){
  S->value++
  if(S->value<0)wakeup(S->list)
}
```

▼ 生产消费者

有一群生产者在生产产品,提供给消费者去消费

在两者之间放入n个缓冲区,生产者把生产的产品放入缓冲区,消费者从缓冲区取出消费

不允许消费者到空缓冲区取

也不允许生产者向满的缓冲区存产品

每个时刻只允许一个生产者或者消费者存或取一个产品

设缓冲区编号0-(n-1)

in-生产者指针

out-消费者指针

初始值都为0

三个信号量

full 放有产品的缓冲区数,初始值为0

empty 表示可供使用的缓冲区数,其初始值为n

mutex 互斥信号量,初值1,表示各进程互斥进入临界区,保证任何时候只有一个进程使用缓冲区

```
int in = 0,out = 0;
semaphore mutex = 1,empty = N,full = 0

生产者进程Producer:
while(TRUE){
  P(empty);
  P(mutex);
  产品送往buffer(in);
  in = (in + 1) mod N
  V(mutex);
  V(full);//增加一个full
}

消费者进程Consumer:
while(TRUE){
  P(full);
  P(mutex);
  从buffer(out)中取出产品
  out = (out +1)mod N
  V(mutex);
  V(empty);//增加一个空位
}


这两个能交换顺序吗?
  P(empty);
  P(mutex);
先抢到使用缓冲区的权利
  P(mutex);
消费者进去p(mutex)无法使用,则无法取出,则一直满
```

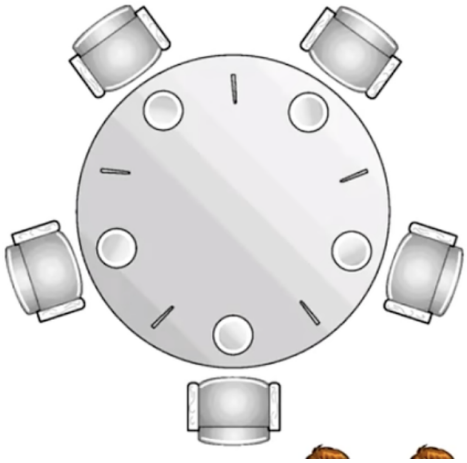
```
  P(empty);


这两个能交换顺序吗?可以
  V(mutex);
  V(full);
```

pv成堆出现

多个p次序不能弟拿到,先对资源信号量操作P(empty)再对互斥信号量操作P(mutex)

▼ 哲学家就餐问题

由Dijkstra提出并解决的哲学家进餐问题是典型的同步问题。
问题描述：有五个哲学家共用一张圆桌，分别坐在周围的五张椅子上，在圆桌上有五个碗和五支筷子，他们的生活方式是交替地进行思考和进餐。平时，一个哲学家进行思考，饥饿时便试图取用其左右最靠近他的筷子，只有在他拿到两只筷子时才能进餐。进餐毕，放下筷子继续思考。



筷子为临界资源

一段时间只允许一个哲学家使用

semaphore chopstick[5] = {1,1,1,1,1}

有可能死锁,每个人都拿起一个筷子

第i位哲学家的活动为

```
do{
  wait(chopstick[i]);
  wait(chopstick[(i+1)%5]);
  ```
  //吃饭
  signal(chopstick[i])
  signal(chopstick[(i+1)%5])
  ```
  //思考
}while(TRUE)
```

解决方案

1,至多允许4个哲学家拿左边的快递,最终保证至少一个哲学家能够进餐,用完后释放两个筷子,然后使更多哲学家用餐

```
semaphore chopstick[5] = {1,1,1,1,1}
semaphore count = 4

do{
  wait(count)
  wait(chopstick[i]);
  wait(chopstick[(i+1)%5]);
  ```
  //吃饭
  signal(chopstick[i])
  signal(chopstick[(i+1)%5])
  signal(count)
  ```
  //思考
}while(TRUE)
```

2,方案2：规定奇数号哲学家先拿他左边的筷子，然后再去拿右边的筷子，而偶数号的相反。

> 例如：1、2号哲学家竞争1号筷子：3、4号哲学家竞争3号筷子（竞争奇数号），获得后再去竞争偶数号筷子，肯定能有一位哲学家获得两只筷子进餐。

```
do{
  if(i%2==1){
    wait(chopstick[i]);
    wait(chopstick[(i+1)%5]);
  }else{
    wait(chopstick[(i+1)%5]);
    wait(chopstick[i]);
```

```
  }

  ```
  //吃饭
  signal(chopstick[i])
  signal(chopstick[(i+1)%5])
  ```
  //思考
}while(TRUE)
```

方案3：仅当哲学家的左右两只筷子均可用时，才允许他拿起筷子进餐。
方案4：对後子进行编号，规定哲学家先取编号小的筷子。
方案5：如果哲学家拿起左边的筷子后，申请右边的筷子得不到满足，则放下左边的筷子，隔一段时间后再申请左边的筷子。

▼ 读者写者

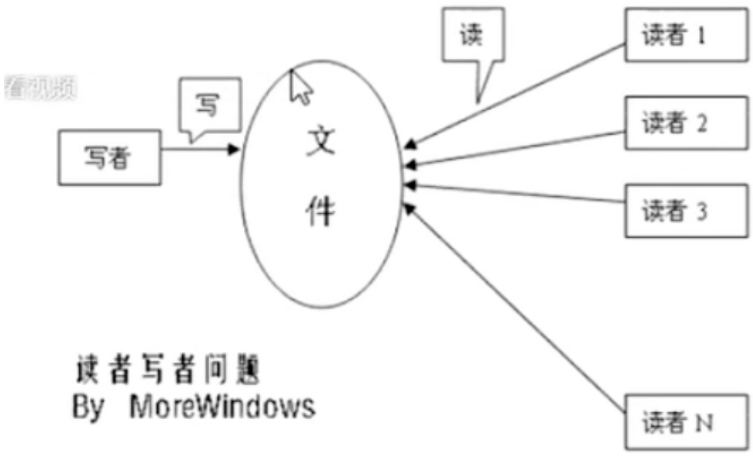
类似文件读写问题,一个数据文件或记录可以被多个进程共享,只读文件的进程为reader,可写的为writer进程

要求

允许多个reader进程同时读(因为读操作不会让数据文件混乱)

不允许writer进程,reader进程同时操作

不允许多个writer进程同时操作



读写不对称

信号量设置

w mutex: 互斥量,写者和其他读者/ 写者互斥地访问 初始值为1

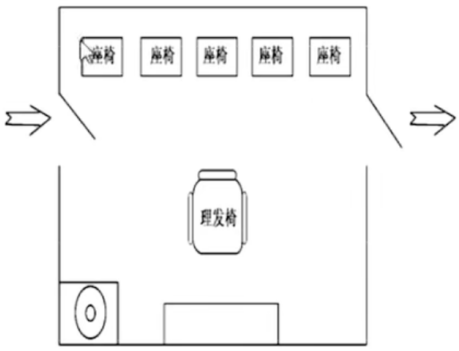
mutex:互斥量,互斥访问临界资源readcount,初始值为1

readcount:读者计数,初值为0

```
读者Readers
while(TRUE){
  P(mutex)
  readcount++
  if(readcount == 1)
    P(wmutex)
  V(mutex)

  执行读操作

  P(mutex)
  readcount--
  if(readcount==0)
    V(wmutex)
  V(mutex)
}

写者Writers
while(TRUE){
  P(wmutex)
  执行写操作
  V(wmutex)
}
```

▼ 理发师

理发店有一名理发师，一把理发椅和几把座椅，等待理发者可坐在上面。
如果没有顾客到来，理发师就坐在理发椅上打盹。
当顾客到来时，就唤醒理发师。如果顾客到来时理发师正在理发，该顾客就坐在椅子上
排队：如果满座了，他就离开这个理发店，到别处去理发。



理发师和每一位顾客各一个进程。
• 理发师进程分析：理发师开始工作时，先看一下店内有无顾客，如果没有，就打瞌睡（阻塞）：如果有，就被唤醒，进行理发，且等待人数减1。注：理发师由于不停地理发故使用while循环。
顾客进程分析：每一位顾客一个进程，故有多个顾客进程。每个进程中，如果顾客数超过座位数，就结束进程。否则等待理发师。顾客理完发（或无座位）就走，因此不需要while循环

顾客队列有上线,生产者消费者的产品可以生产1000个,只要排队进来就行,可以是无穷,但是,满座后,顾客离开店不在排队,不需要pv

引入了个信号量和一个控制变量：

- 控制变量waiting 用来记录等候理发的顾客数，初值为0：
- 信号量customers 用来记录等候理发的顾客数，并用作阻塞理发师进程，初值为0；
- 信号量barbers 用来记录正在等候顾客的理发师数，并用
  作阻塞顾客进程，初值为0：
- 信号量mutex 用于互斥，初值为1。
-

```
void barber(void){
  while(true){
    P(customers)///如果没有顾客,则理发师打瞌睡
    P(mutex)//互斥进入临界区
    waiting--
    V(barbers)//一个理发师准备理发
    V(mutex)//退出临界区
```

```
    cut_hair()//理发

  }
}

void customer(void){
  P(mutex)//互斥进入临界区
  if(waiting < CHARIS){//超过上限
    waiting++
    V(customers)//若有必要,唤醒理发师
    V(mutex)//推出临界区
    P(barbers)//如果理发师正忙,则顾客打瞌睡
    get_haircut()
}else
    V(mutex)//店里满人了,不等了
}
```