进程同步与信号量习题课

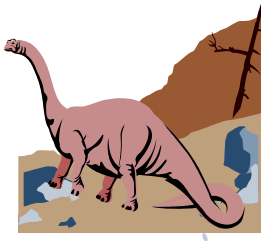
肖 卿 俊
办公室：九龙湖校区计算机楼212室
电邮：csqjxiao@seu.edu.cn
主页： https://csqjxiao.github.io/PersonalPage
电话：025-52091022

进程同步与互斥：习题1

- 现有4个进程R1，R2，W1，W2，它们共享可以存放一个数的缓冲器B。
  - 进程R1每次把从键盘上输入的一个数存放到缓冲器B中，供进程W1打印输出；
  - 进程R2每次从磁盘上读一个数放到缓冲器B中，供进程W2打印输出。
  - 当一个进程把数据存放到缓冲器B后，在该数还没有被打印输出之前不准任何进程再向缓冲器中存数
  - 在缓冲器B中还没有存入一个新的数之前不允许任何进程加快从缓冲区中取出打印。
- 怎样才能使这四个进程并发执行时协调工作？

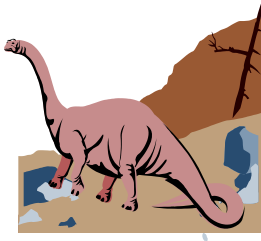
进程同步与互斥：习题1

- 这四个进程实际上是
  - 两个生产者 R1，R2，和
  - 两个消费者 W1，W2，
- 各自生成不同的产品给各自的消费对象。
- 他们共享一个的缓冲器。由于缓冲器只能存放一个数，所以，R1和R2在存放数时必须互斥。而R1和W1、R2和W2之间存在同步。



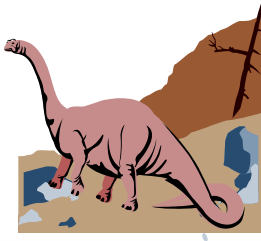
进程同步与互斥：习题1

- 为了协调它们的工作可定义三个信号量：
- empty：表示能否把数存人缓冲器B，初始值为1.
- full1：表示R1是否已向缓冲器存入从键盘上读入的一个数，初始值为0.
- full2：表示R2是否已向缓冲器存入从磁盘上读入的一个数，初始值为0.



进程同步与互斥：习题1

- 算法



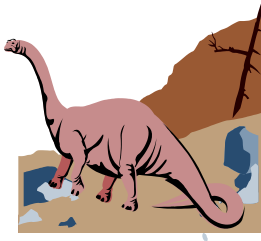
进程同步与互斥：习题1

semaphore empty=1, full1=full2=0;

buffer B;

```
process R1() {
    int x;
    do {
        接收来自键盘的数;
        x=接收的数;
        wait(empty);
        B=x;
        signal(full1);
     } while(1);
}
```

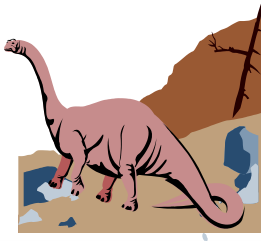
```
process R2() {
    int y;
    do {
        从磁盘上读一个数;
        y=接收的数;
        wait(empty);
        B=y;
        signal(full2);
     } while(1);
}
```



进程同步与互斥：习题1

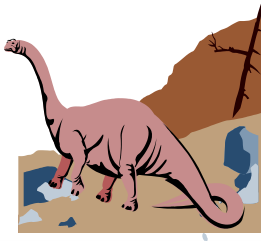
```
process W1() {
    int i;
    do {
        wait(full1);
        i=B;
        signal(empty);
        打印i中数;
    } while(1);
}
```

```
process W2() {
    int j;
    do {
        wait(full2);
        j=B;
        signal(empty);
        打印j中数;
    } while(1);
}
```



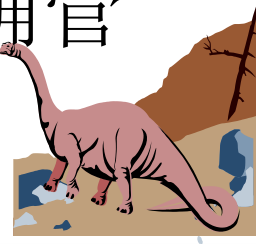
随堂练习

- 如果有三个进程R、W1、W2共享一个缓冲器B，而B中每次只能存放一个数。
- 当缓冲器中无数时，进程R可以将从输入设备上读入的数存放到缓冲器。只有等缓冲区中的数被取出后，进程R才能再存放下一个数。
  - 若存放到缓冲器中的是奇数，则允许进程W1将其取出打印。
  - 若存放到缓冲器中的是偶数，则允许进程W2将其取出打印。
  - 进程W1或W2对每次存入缓冲器的数只能打印一次；W1和W2都不能从空缓冲中取数。
- 写出这三个并发进程能正确工作的程序。



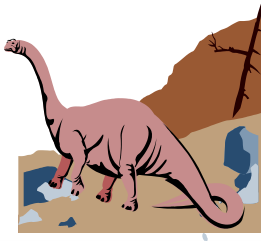
进程同步与互斥：习题2

- a、b两点之间是一段东西向的单车道，现要设计一个自动管理系统，管理规则如下：
  - 当ab之间有车辆在行驶时，同方向的车可以同时驶入ab段，但另一方向的车必须在ab段外等待。
  - 当ab之间无车辆在行驶时，到达a点（或b点）的车辆可以进入ab段，但不能从a点和b点同时驶入
  - 当某方向在ab段行驶的车辆驶出了ab段且暂无车辆进入ab段时，应当让另一方向等待的车辆进入ab段行驶。
- 请用信号量机制为工具，对ab段实现正确管理以保证行驶安全。



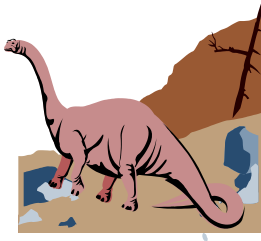
进程同步与互斥：习题2

- 这是读者—写者问题的一种变形。
- 我们设置两个共享变量和三个互斥信号量
  - 变量ab记录当前ab段上由a点进入的车辆的数量，
  - mutexab用于从a点进入的车互斥访问共享变量ab
  - 变量ba记录当前ab段上由b点进入的车辆的数量，
  - mutexba用于从b点进入的车互斥访问共享变量ba
  - mutex用于a、b点的车辆互斥进入ab段。
- 两个共享变量ab和ba的初值分别为0、0。
- 三个信号量的初值分别为1、1和1。



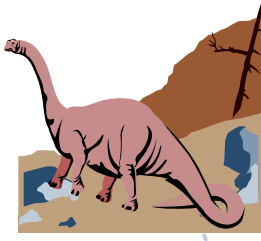
进程同步与互斥：习题2

- 算法



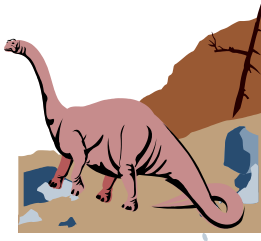
进程同步与互斥：习题2

```
semaphore mutexab=1, mutexba=1, mutex=1;     int ab=ba=0;
process Pab () {                             process Pba () {
    do {                                         do {
        wait(mutexab);                               wait(mutexba);
        if(ab==0) wait(mutex);                       if(ba==0) wait(mutex);
        ab=ab+1;                                     ba=ba+1;
        signal(mutexab);                             signal(mutexba);
        车辆从a点驶向b点;                            车辆从b点驶向a点;
        wait(mutexab);                               wait(mutexba);
        ab=ab-1;                                     ba=ba-1;
        if(ab==0) signal(mutex);                     if(ba==0) signal(mutex);
        signal(mutexab);                             signal(mutexba);
    } while(1);                                  } while(1);
}                                            }
```



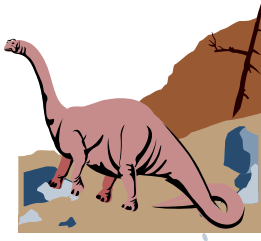
进程同步与互斥：习题3

- 和尚挑水问题：寺庙里有多个小、老和尚，一水缸，水缸容积10桶水。
- 小和尚取水放入水缸，老和尚从水缸取水饮用
- 水取自同一水井，水井每次只容一个桶取水。
- 桶总数3个，每次倒水入水缸水、或者从水缸中取水都仅为一桶。
- 试用信号量描述和尚取水、饮水的互斥与同步过程。



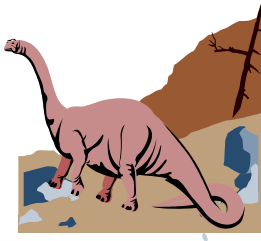
进程同步与互斥：习题3

- semaphore mutex_well = 1; // 水井互斥体
- semaphore mutex_tank = 1; // 水缸互斥体
- semaphore tank_empty = 10; // 水缸的空位数
- semaphore tank_full = 0; // 水缸目前的水量
- semaphore buckets = 3; // 水桶资源个数



进程同步与互斥：习题3

- 算法



老和尚程序：

begin

wait(buckets);
wait(tank_full);
wait(mutex_tank);

用桶从水缸取水；

signal(mutex_tank);
signal(tank_empty);
signal(buckets);

end

小和尚程序：

begin

wait(buckets);
wait(mutex_well);

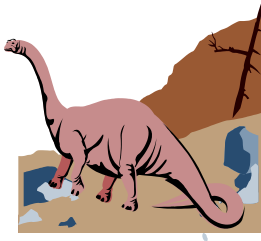
用桶从水井打水；

signal(mutex_well);
wait(tank_empty);
wait(mutex_tank);

用桶往缸中放水；

signal(mutex_tank);
signal(tank_full);
signal(buckets);

Is this solution correct? Why?



进程同步与互斥：习题3

- 因为老和尚的**wait(buckets)**操作放在**wait(tank_full)**之前，可能会发生如下死锁现象
  - 水缸为空的时候，三个老和尚同时试图取水喝。这样所有的水桶资源都会被老和尚们占用，而小和尚们就拿不到水桶去井里取水，所有进程被互相锁住

- 相类似，小和尚打水时候，因为**wait(buckets)**放在**wait(tank_empty)**之前，可能出现下面情况
  - 水缸为满的时候，三个小和尚同时尝试打水，这样所有的水桶资源被小和尚们占用，而老和尚就拿不到水桶到水缸取水喝，所有进程被互相锁住

老和尚程序：

begin

wait(tank_full);
wait(buckets);
wait(mutex_tank);

用桶从水缸取水；

signal(mutex_tank);
signal(buckets);
signal(tank_empty);

end

小和尚程序：

begin

wait(tank_empty);
wait(buckets);
wait(mutex_well);

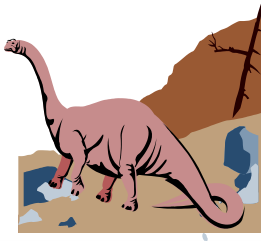
用桶从水井打水；

signal(mutex_well);
wait(mutex_tank);

用桶往缸中放水；

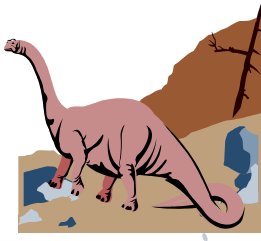
signal(mutex_tank);
signal(buckets);
signal(tank_full);

end



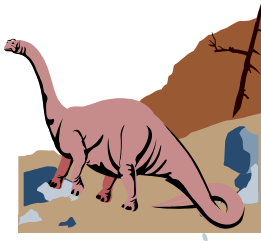
进程同步与互斥：习题4

- 假设有两个生产者进程A、B和一个销售者进程C，他们共享一个无限大的仓库。
  - 生产者每次循环生产一个产品，然后入库供销售。
  - 销售者每次循环从仓库中取出一个产品进行销售。
  - 不允许同时入库，也不允许边入库边出库。
  - 要求生产A产品和B产品的件数满足以下关系：
    $-n \le$ A的件数 - B的件数 $\le m$，其中n、m是正整数
  - 要求销售A产品和B产品的件数满足以下关系：
    $-n \le$ A的件数 - B的件数 $\le m$，其中n、m是正整数
- 试用信号量实现这一同步问题。



进程同步与互斥：习题4

- 分析：本题中存在着以下的同步和互斥关系：
  - 生产者A、B和消费者之间不能同时将产品入库和出库，故仓库是一个临界资源；
  - 两个生产者之间必须进行同步，当生产的A、B产品的件数之差大于等于m时，生产者A必须等待；小于等于-n时，生产者B必须等待；



进程同步与互斥：习题4

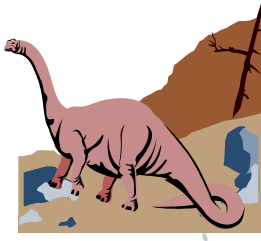
- 生产者和销售者之间也必须进行同步
  - 销售者想销售A产品时，仓库里必须有A才能进行销售；类似的，销售者想销售B产品时，仓库里必须有B才能进行销售；
- 销售者自己也需要保持同步步调
  - 由于销售的产品件数必须满足关系

    $-n \le$ A的件数－B的件数 $\le m$，

    因此当销售的A、B产品件数之差大于等于m而仓库中已无B产品时，或者销售的A、B产品的件数之差小于等于-n而仓库中已无A产品时，销售者C必须等待

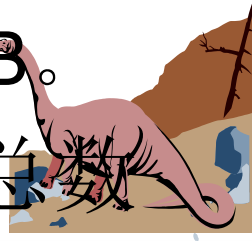
进程同步与互斥：习题4

- 为了互斥地入库和出库，为仓库设置互斥信号量mutex，其初值为1。
- 由于仓库无限大，入库只要操作互斥就可以完成，无须考虑仓库是否有空位。
- 但出库要考虑有无产品，因此需要设置两个资源信号量，其中
  - SA对应于仓库中的A产品量，
  - SB对应于仓库中的B产品量。
- 它们的初值都为0.



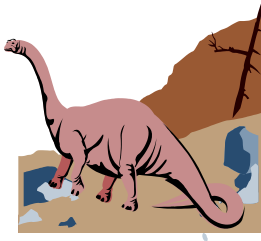
进程同步与互斥：习题4

- 为了使生产的产品件数满足-n ≤ A的件数－B的件数 ≤ m，须设置两个同步的信号量，其中
  - SAB表示当前允许A生产的产品数量，初值为m，
  - SBA表示当前允许B生产的产品数量，初值为n；
- 为了让销售满足：-n ≤ A的件数－B的件数 ≤ m，用difference变量表示所销售的A、B产品数量之差, 即difference = A的件数－B的件数;
  - 当difference==-n时，不能取产品B，只能取A；
  - 当difference==m时，不能取产品A，只能取B；
  - 当-n<difference<m时，既可以取A也可以取B。
- 还需设置信号量S，对应于仓库中的产品总数。



进程同步与互斥：习题4

- 算法



进程同步与互斥：习题4

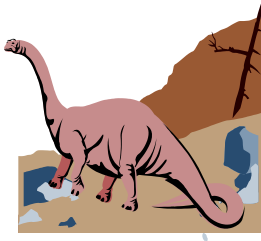
```
Semaphore  SAB=m, SBA=n, S=0, SA=0, SB=0, mutex=1;
process producerA( ) {
    do {//生产产品，要保证 -n≤A的件数-B的件数≤m
        wait(SAB);
        produce a product A;
        signal(SBA);
        wait(mutex); //入库操作，满足出入库操作互斥即可
        add the product A to the storehouse;
        signal(mutex);
        signal(SA); //入库产品A一件，所以给SA增值
        signal(S);  //给S增值，因为S是仓库中全部产品的数量
    } while(1);
}
```

进程同步与互斥：习题4

```
process producerB( ) {
    do {//生产产品，要保证 -n≤A的件数-B的件数≤m
        wait(SBA);
        produce a product B;
        signal(SAB);
        wait(mutex); //入库操作，满足出入库操作互斥即可
        add the product B to the storehouse;
        signal(mutex);
        signal(SB); //入库产品B一件，所以给SB增值
        signal(S); //给S增值，因为S是仓库中全部产品的数量
    } while(1);
}
```

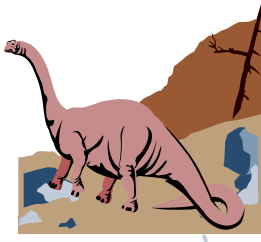
进程同步与互斥：习题4

```
process consumer( ) {
    do {
        wait(S);  //首先检查有无产品，无产品阻塞，有产品，下面操作将会取走一件产品，所以S减1
        if(difference <= -n) {
            wait(SA); // difference<=-n时只能取A产品一件
            //出库操作，满足出入库操作互斥
            wait(mutex);
            take a product A from storehouse;
            signal(mutex);
            difference++; //取A产品一件，difference++
        }
```



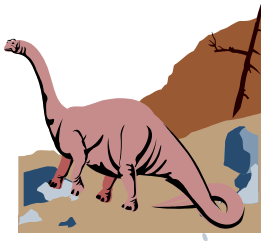
进程同步与互斥：习题4

```
else if(difference>=m) {
    wait(SB); // difference>=m时只能取B产品一件
    //出库操作，满足出入库操作互斥
    wait(mutex);
    take a product B from storehouse;
    signal(mutex);
    difference--; //取B产品一件，difference--
} else { //-n<difference<m，既可以取产品A也可以取产品B，随意取一件产品，之后再根据取得产品是A还是B做处理
    //出库操作，满足出入库操作互斥
    wait(mutex);
    take a product from storehouse;
    signal(mutex);
```

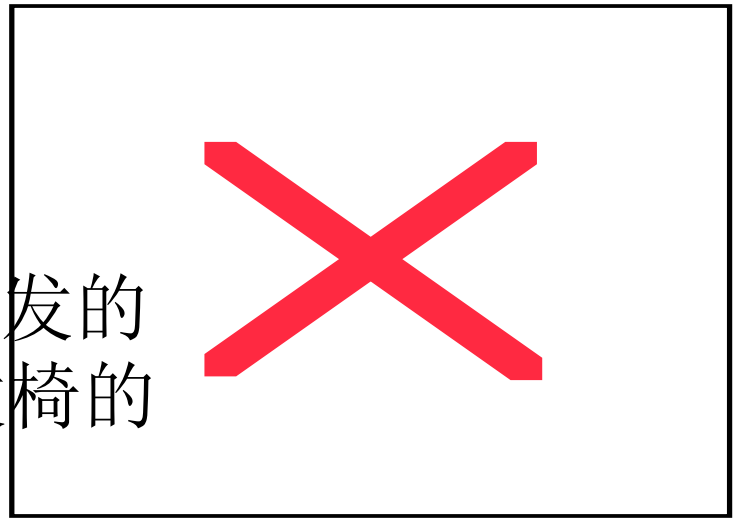


进程同步与互斥：习题4

```
        if(product_type == A) {//取的是产品A，则信号量SA减1，这里不可能发生没有A产品，进程C需要阻塞的情况
            wait(SA);
            difference++; //取A产品一件，difference++
        } else {//取的是产品B，则信号量SB减1，这里不可能发生没有B产品，进程C需要阻塞的情况
            wait(SB);
            difference--; //取B产品一件，difference--
        }
    }
    sell the product;
  } while(1);
}
```



习题5



- 嗜睡的理发师问题：
  - 一个理发店由一个有N张沙发的等候室和一个放有一张理发椅的理发室组成。
  - 没有顾客要理发时，理发师便去睡觉。
  - 当一个顾客走进理发店时，如果理发师因无顾客正在睡觉，则由新到的顾客唤醒理发师为其理发；否则，如果理发师正在为其他顾客理发，则该顾客就找一张空沙发坐下等待；如果所有的沙发都已经被先到的顾客占用，他便离开理发店。
  - 在理发完成后，顾客必须支付费用，等到理发师收取费用并给予回执之后才能离开理发店。
- 试用信号量实现这一同步问题。

进程同步与互斥：习题5

- 分析：本题中，顾客进程和理发师进程之间存在着多种同步关系：

1. 只有在理发椅空闲时，顾客才能做到理发椅上等待理发师理发，否则顾客便必须等待；只有当理发椅上有顾客时，理发师才可以开始理发，否则他也必须等待；这种同步关系类似于单缓冲的生产者-消费者问题中的同步关系，故可通过信号量empty和full来控制。

进程同步与互斥：习题5

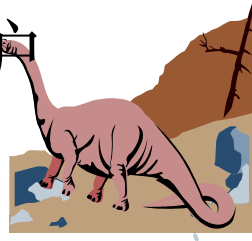
2. 理发师为顾客理发时，顾客必须等待理发的完成，并在理发完成后理发师唤醒他，这可单独使用一个信号量cut来控制。
3. 顾客理完发后必须向理发师付费，并等理发师收费后顾客才能离开；而理发师则需等待顾客付费，并在收费后唤醒顾客以允许他离开，这可分别通过两个信号量payment和receipt来控制。
4. 等候室中的N张沙发是顾客竞争的资源，故还需为它们设置了一个资源信号量sofa。

进程同步与互斥：习题5

5. 为了控制顾客的人数，使顾客能够在所有沙发都被占用时离开理发店，还必须设置一个整型变量count来统计理发店里面的顾客数

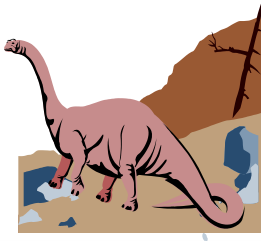
6. 最好count计数既包括正在理发的顾客，也包括沙发上等待的顾客。这样当顾客发现自己是店里第一个顾客时，可以直接上理发椅

7. 该count变量将被多个顾客进程访问并修改，需要通过一个互斥信号量mutex来保护



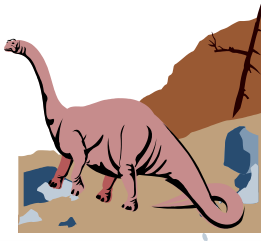
进程同步与互斥：习题5

- 为解决上述问题，需设置一个整型变量count用来对理发店里的顾客进行计数，并需设置7个信号量，其中：
  - mutex用来实现顾客进程对count变量的互斥访问，其初值为1；
  - sofa对应于等候室中的N张沙发，其初值为N；
  - empty表示是否有空闲的理发椅，其初值为1；
  - full表示理发椅上是否有等待理发的顾客，初值为0；
  - cut用来等待理发的完成，其初值为0；
  - payment表示用来等待付费，其初值为0；
  - receipt用来等待收费，其初值为0。



进程同步与互斥：习题5

- 算法：

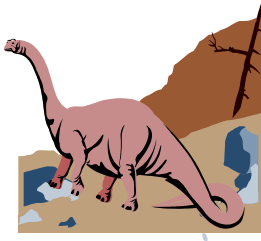


进程同步与互斥：习题5

int count=0; // 理发店中的顾客数，正在理发的加沙发上等待的

semaphore mutex=1, sofa=N, empty=1, full=0, cut=0, payment=0, receipt=0;

```
process barber( ) {  //理发师进程
    do {
        wait(full);    //测试理发椅上是否有顾客，无则阻塞
        cut hair;
        signal(cut);    //告知顾客理发完成
        wait(payment);  //测试顾客是否付费
        accept payment;
        signal(receipt);  //告知顾客收费完毕
    } while(1);
}
```



进程同步与互斥：习题5

process guest(int i) { // 第i个顾客进程

wait(mutex);

if(count>N) { // 沙发中已经坐满客人N个和正在理发的1位，没有沙发，客人直接离开理发店

signal(mutex);

exit shop and return;

}

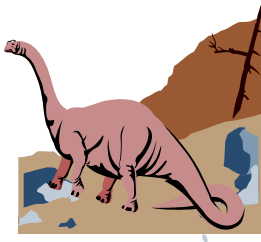
count = count + 1;

if(count == 1) { //意味着这个店里只有这一个顾客，他不必要申请沙发，直接申请理发椅即可

signal(mutex) ;

wait(empty);

}



进程同步与互斥：习题5

else { //意味着原来理发店里有客人，所以要坐到沙发上等待

signal(mutex) ;

wait(sofa); //申请沙发，肯定能得到

sit on sofa;

wait(empty); //申请理发椅

get up from sofa;

signal(sofa); //归还沙发资源

}

// 在理发椅上的理发过程

sit on the barber chair;

signal(full); //若理发师阻塞，则唤醒理发师，若未阻塞，则将full由0增1，理发师申请理发时即可通过

wait(cut); //测试理发是否完成，没有则阻塞

进程同步与互斥：习题5

```
// 理发结束后的支付过程
pay;
signal(payment); //若理发师阻塞，则唤醒理发师告知其已经付费，若未阻塞，则将paymentl由0增1，理发师申请理要求付费时即可通过
wait(receipt); // 测试理发师收费是否完成，没有则阻塞
get up from the baber_chair; // 收费完成，顾客离开理发椅
signal(empty); //释放理发椅
wait(mutex); //对count 临界资源操作，用mutex完成互斥
count = count - 1;
signal(mutex);
exit shop;
}
```