2009：缓冲区问题

- **题目描述**

三个进程 P1、P2、P3互斥使用一个包含 $N(N>0)$个单元的缓冲区。P1每次用 produce()生成一个正整数并用put()送入缓冲区某一空单元中；P2 每次用 getodd()从该缓冲区中取出一个奇数并用 countodd()统 计奇数个数；P3每次用 geteven()从该缓冲区中取出一个偶数并用 counteven()统计偶数个数。请用信号量机 制实现这三个进程的同步与互斥活动，并说明所定义信号量的含义。要求用伪代码描述。

- **实现**

定义信号量 odd 控制 P1与 P2之间的同步；even 控制 P1与 P3之间的同步；empty 控制生产者与消费者之间的同步；mutex 控制进程间互斥使用缓冲区。程序如下：

```
semaphore odd = 0, even = 0, empty = N, mutex = 1;
P1( )
{
    x = produce();  //生成一个数
    P(empty);       //判断缓冲区是否有空单元
    P(mutex);       //缓冲区是否被占用
    Put();
    V(mutex);       //释放缓冲区
    if(x%2 == 0)
        V(even);    //如果是偶数，向 P3 发出信号
    else
        V(odd);     //如果是奇数，向 P2 发出信号
}
P2( )
{
    P(odd);         //收到 P1 发来的信号，已产生一个奇数
    P(mutex);       //缓冲区是否被占用
    getodd();
    V(mutex);       //释放缓冲区
    V(empty);       //向 P1 发信号，多出一个空单元
    countodd();
}
P3( )
{
    P(even);        //收到 P1 发来的信号，已产生一个偶数
    P(mutext);      //缓冲区是否被占用
    geteven();
    V(mutex);       //释放缓冲区
    V(empty);       //向 P1 发信号，多出一个空单元
    counteven();
}
```

2011：顾客问题

- **题目描述**

某银行提供 1 个服务窗口和 10 个供顾客等待的座位。顾客到达银行时，若有空座位，则到取号机上领取一个号，等待叫号。取号机每次仅允许一位顾客使用。当营业员空闲时，通过叫号选取一位顾客，并为其服务。顾客和营业员的活动过程描述如下：

```
cobegin
{
    process 顾客 i
    {
        从取号机获取一个号码;
        等待叫号;
        获取服务;
    }
    process 营业员
    {
        while（TRUE）
        {
            叫号;
            为客户服务;
        }
    }
}coend
```

请添加必要的信号量和 P、V（或 wait()、signal()）操作，实现上述过程中的互斥与同步。要求写出完整的过程，说明信号量的含义并赋初值。

- **实现**

```
semaphore seets = 10,    // 有 10 个坐位的资源信号量
          mutex = 1,     // 取号机互斥信号量
     haveCustom = 0;     // 顾客与营业员同步，无顾客时营业员休息
process 顾客
{
    P(seets);            // 等空位
    P(mutex);            // 申请使用取号机
    从取号机上取号;
    V(mutex);            // 取号完毕
    V(haveCustom);       // 通知营业员有新顾客到来
    等待营业员叫号;
    V(seets);            // 离开坐位
    接受服务;
}
process 营业员
{
    while(True)
    {
        P(haveCustom); // 没有顾客则休息
        叫号;
        为顾客服务;
    }
}
```

2013：游客参观问题

- **题目描述**

某博物馆最多可容纳 500 人同时参观，有一个出入口，该出入口一次仅允许一个人 通过。参观者的活动描述如下：

```
cobegin
参观者进程 i：
{
    ...
    进门
    ...
    参观
    ...
    出门
    ...
}
coend
```

请添加必要的信号量和 P、V（或 wait（）、signal（））操作，以实现上述过程中的互斥 与同步。要求写出完整的过程，说明信号量的含义并赋初值。

- **实现**

```
定义两个信号量
Semaphore empty = 500;       //博物馆可以容纳的最多人数（2 分）
Semaphore mutex = 1;         //用于出入口资源的控制（2 分）
参观者进程 i;
{
    ...
    P(empty);
    P(mutex);
    进门;
    V(mutex);
    参观;
    P(mutex);
    出门;
    V(mutex);
    V(empty);
    ...
}
```

2014：生产者消费者问题[重]

- **题目描述**

系统中有多个生产者进程和多个消费者进程，共享一个能存放 1000 件产品的环形缓冲区（初始为空）。当缓冲区未满时，生产者进程可以放入其生产的一件产品，否则等待； 当缓冲区未空时，消费者进程可以从缓冲区取走一件产品，否则等待。要求一个消费者进程从缓冲区连续取出 10 件产品后，其他消费者进程才可以取产品。请使用信号量 P，V(wait()， signal())操作实现进程间的互斥与同步，要求写出完整的过程，并说明所用信号量的含义和初值。

- **实现**

设置四个变量 mutex1、mutex2、empty 和 full，mutex1 用于一个控制一个消费者进程一个周期(10 次)内对于缓冲区的控制，初值为 1，mutex2 用于进程单次互斥的访问缓冲区， 初值为 1，empty 代表缓冲区的空位数，初值为 0，full 代表缓冲区的产品数，初值为 1000， 具体进程的描述如下：

```
semaphore mutex1=1;
semaphore mutex2=1;
semaphore empty=n;
semaphore full=0;
producer(){
    while(1){
        生产一个产品;
        P(empty);              //判断缓冲区是否有空位
        P(mutex2);             //互斥访问缓冲区
        把产品放入缓冲区;
        V(mutex2);             //互斥访问缓冲区
        V(full);               //产品的数量加 1
    }
}
consumer(){
    while(1){
        P(mutex1)              //连续取 10 次
        for(int i = 0; i <= 10; ++i){
            P(full);           //判断缓冲区是否有产品
            P(mutex2);         //互斥访问缓冲区
            从缓冲区取出一件产品;
            V(mutex2);         //互斥访问缓冲区
            V(empty);          //腾出一个空位
            消费这件产品;
            }
        V(mutex1)
    }
}
```

2015：辩论问题

- **题目描述**

有A、B两人通过信箱进行辩论，每个人都从自己的信箱中取得对方的问题，将答案和向对方提出的新问题组成一个邮件放入对方的信箱中。假设A的信箱最多放M个邮件，B的信箱最多放N个邮件。初始时A的信箱中有x个邮件($0 < x < M$)，B的信箱中有y个邮件($0 < y < N$)。辩论者每取出一个邮件，邮件数减1。A和B两人的操作描述如下：

CoBegin

A {
while(TRUE){
从 A 的信箱中取出一个邮件;
回答问题并提出一个新问题;
将新邮件放入 B 的信箱;
}
}
	B {
	while(TRUE){
	从 B 的信箱中取出一个邮件;
	回答问题并提出一个新问题;
	将新邮件放入 A 的信箱;
	}
	}

CoEnd

当信箱不为空时，辩论者才能从信箱中取邮箱，否则等待。当信箱不满时，辩论者才能将新邮件放入信箱，否则等待。请添加必要的信号量和P、V(或wait、signal)操作，以实现上述过程的同步。要求写出完整的 过程，并说明信号量的含义和初值。

- **实现**

```
semaphore Full_A = x ;                  //Fuu_A表示A的信箱中的邮件数量
semaphore Empty_A = M-x;                //Empty_A表示A的信箱中还可存放的邮件数量
```

```
semaphore Full_B = y ;                  //Full_B表示B的信箱中的邮件数量
semaphore Empty_B = N-y;                //Empty_B表示B的信箱中还可存放的邮件数量
semaphore mutex_A = 1 ;                 //mutex_A用于A的信箱互斥
semaphore mutex_B = 1 ;                 //mutex_B用于B的信箱互斥
A{
    while(TRUE){
        P(Full_A);
        P(mutex_A);
        从 A 的信箱中取出一个邮件
        V(mutex_A);
        V(Empty_A);
        回答问题并提出一个新问题
        P(Empty_B);
        P(mutex_B);
        将新邮件放入 B 的信箱
        V(mutex_B);
        V(Full_B);
    }
}
B{
    while(TRUE){
        P(Full_B);
        P(mutex_B);
        从 B 的信箱中取出一个邮件
        V(mutex_B);
        V(Empty_B);
        回答问题并提出一个新问题
        P(Empty_A);
        P(mutex_A);
        将新邮件放入 A 的信箱
        V(mutex_A);
        V(Full_A);
    }
}
```

2017：并发线程问题

- **题目描述**

某进程中有3个并发执行的线程thread1、thread2和thread3，其伪代码如下所示：

```
//复数的结构类型定义
typedef struct
}
    float a;
    float b;
}   cnum;
cnum x，y，z；//全局变量

//计算两个复数之和
cnum add(cnum p，cnum q)
{
    cnum s;
    s.a=p.a+q.a;
    s.b=p.b+q.b;
    return s;
}
```

```
thread1
{
    cnum w;
    w=add(x，y);
    ……
}

thread2
}
    cnum w;
    w=add(y，z);
    ……
}
```

```
thread3
{
    cnum w;
    w.a=1;
    w.b=1;
    z=add(z，w);
    y=add(y，w);
    ……
}
```

请添加必要的信号量和P、V(或wait( )、signal( ))操作，要求确保线程互斥访问临界资源， 并且最大程度地并发执行。

- **实现**

semaphore mutex_y1=1；//mutex_y1用于thread1与thread3对变量y的互斥访问。

semaphore mutex_y2=1；//mutex_y2用于thread2与thread3对变量y的互斥访问。

semaphore mutex_z=1； //mutex_z用于变量z的互斥访问。(1分)互斥代码如下：

```
thread1
{
    cnum w;
    wait(mutex_y1);
    w=add(x，y);
    signal(mutex_y1);
    ……
}
```

```
thread2
{
    cnum w;
    wait(mutex_y2);
    wait(mutex_z);
    w=add(y，z);
    signal(mutex_z);
    signal(mutex_y2);
    ……
}
```

```
thread3
{
    cnum w;
    w.a=1;
    w.b=1;
    wait(mutex_z);
    z=add(z，w);
    signal(mutex_z);
    wait(mutex_y1);
    wait(mutex_y2);
    y=add(y，w);
    signal(mutex_y1);
    signal(mutex_y2);
    ……
}
```

2019：哲学家问题

- **问题描述**

有 n（n≥3）位哲学家围坐在一张圆桌边，每位哲学家交替地就餐和思考。在圆桌中心有 m（m≥1）个碗，每两位哲学家之间有一根筷子。每位哲学家必须取到一个碗和两侧的筷子后，才能就餐，进餐完毕，将碗和筷子放回原位，并继续思考。为使尽可能多的哲学家同时就餐，且防止出现死锁现象，请使用信号量的 P、V 操作［wait()、signal()操作］描述上述过程中的互斥与同步，并说明所用信号量及初值的含义。

- **实现**

回顾传统的哲学家问题，假设餐桌上有 n 个哲学家、n 根筷子，那么可以用这种方法避免 死锁（王道单科书上提供了这一思路）：限制至多允许 n-1 个哲学家同时"抢"筷子，那么至少会有 1 个哲学家可以获得两根筷子并顺利进餐，于是不可能发生死锁的情况。

本题可以用碗这个限制资源来避免死锁：当碗的数量 m 小于哲学家的数量 n 时，可以直接让碗的资源量等于 m，确保不会出现所有哲学家都拿一侧筷子而无限等待另一侧筷子进而造成死锁的情况；当碗的数量 m 大于等于哲学家的数量 n 时，为了让碗起到同样的限制效果，我们让碗的资源量等于 n-1，这样就能保证最多只有 n-1 个哲学家同时进餐，所以得到碗的资源量 为 min{ n-1 , m}。在进行 PV 操作时，碗的资源量起限制哲学家取筷子的作用，所以需要先对碗的资源量进行 P 操作。具体过程如下：

```
//信号量
semaphore bowl;              //用于协调哲学家对碗的使用
semaphore chopsticks[n];     //用于协调哲学家对筷子的使用
for(int i=0;i<n;i++)
    chopsticks[i]=1;         //设置两个哲学家之间筷子的数量
bowl = min(n-1,m);           //bowl≤n-1，确保不死锁
CoBegin
while(TRUE){                 //哲学家 i 的程序
    思考;
    P(bowl);                 //取碗
    P(chopsticks[i]);        //取左边筷子
    P(chopsticks[(i+1)%n]); //取右边筷子
    就餐;
    V(chopsticks[i]);
    V(chopsticks[(i+1)%n]);
    V(bowl);
}
CoEnd
```