



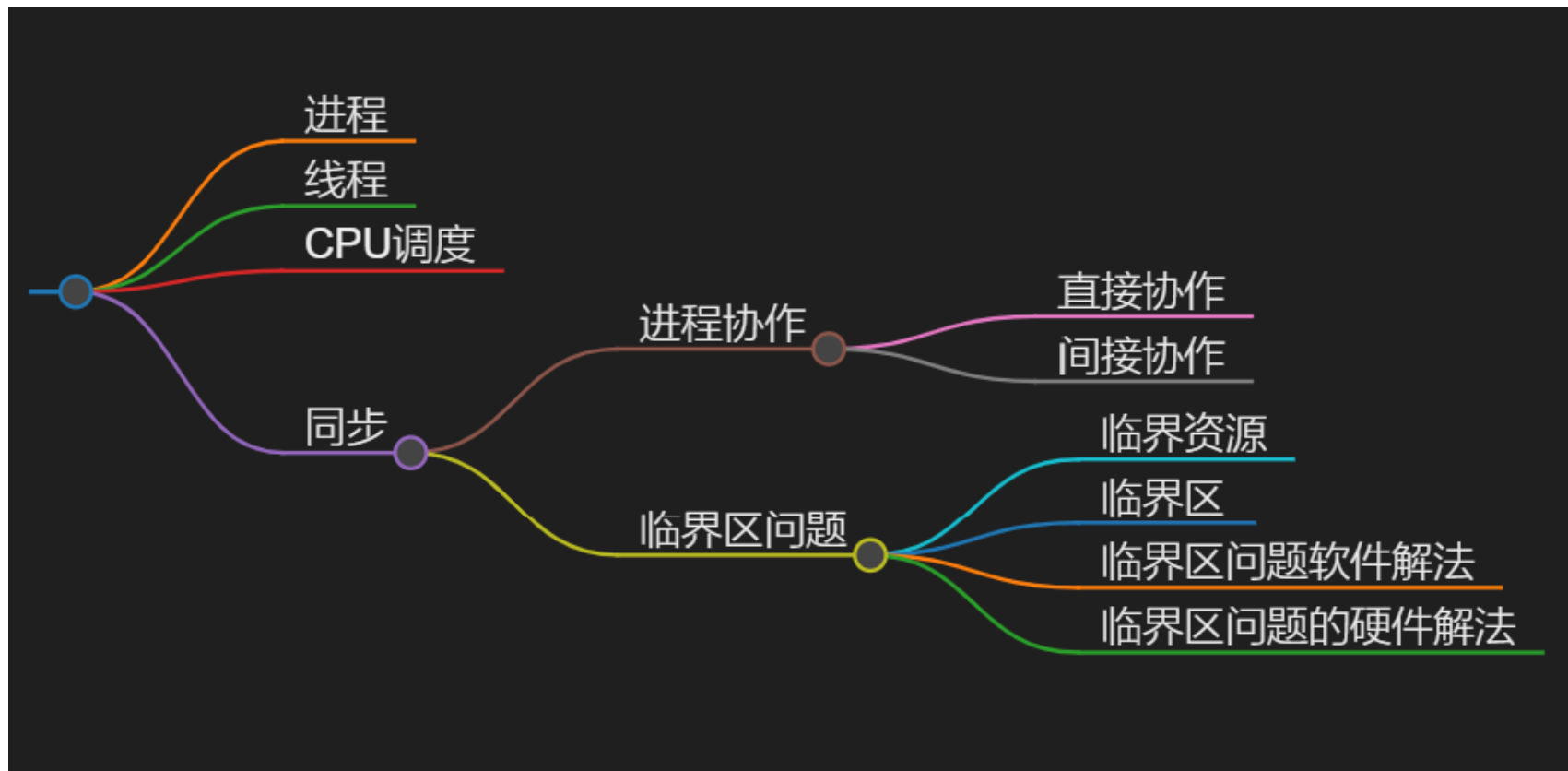
操作系统

L09 进程同步2

胡燕
大连理工大学 软件学院



进程同步 (上)





互斥的三大要求

互斥访问 (Mutual Exclusion)

空闲让进 (Progress)

有限等待 (Bounded Waiting)



互斥的硬件解法

增加TestAndSet指令

```
do{  
    while(TestAndSet(lock));  
    临界区  
    lock = false;  
    剩余区;  
}while(1);
```



如何满足有限等待要求

do{

```
waiting[i] = true;
key = true;
while(waiting[i]&&key)
    key=TestAndSet(lock);
waiting[i]=false;
```

表示进程 P_i 处于
等待获取锁的状态

如果进程 P_i 抢到了锁,
记录 $key=false$

临界区

```
j=(i+1)%n;
while(j!=i && !waiting[j])
    j = (j+1)%n;
if(j==i)
    lock = false;
else
    waiting[j] = false;
```

按照进程编号找到下一个等待
进入临界区的进程
($waiting[j]=true$)

找到等待进入临界区的下一进
程后, 将 $waiting[j]$ 设为false
(可以使其跳出进入区while循
环, 进入临界区)

剩余区

}while(1);

关于临界区问题(Critical Section Problem)是一个算法(假设只有进程P0和P1可能进入该临界区), 算法如下(i为0或1), 该算法_____。

- ☐ A 不能保证进程互斥进入临界区, 且会出现“饥饿”
- ☒ B 不能保证进程互斥进入临界区, 但不会出现“饥饿”
- ☐ C 保证进程能互斥进入临界区, 但会出现“饥饿”
- ☐ D 保证进程互斥进入临界区, 不会出现“饥饿”

```

进程Pi
repeat
  retry:if(turn ≠ -1)turn:=i;
        if(turn ≠ i) go to retry;
  turn:=-1;
  critical section(临界区)
  turn=0;
  remainder section(剩余区)
until false;
    
```

提交

并发进程中与共享变量有关的程序段，称为（ ）。

- ☐ A 公共子程序
- ☒ B 临界区
- ☐ C 治理区
- ☐ D 公共数据区

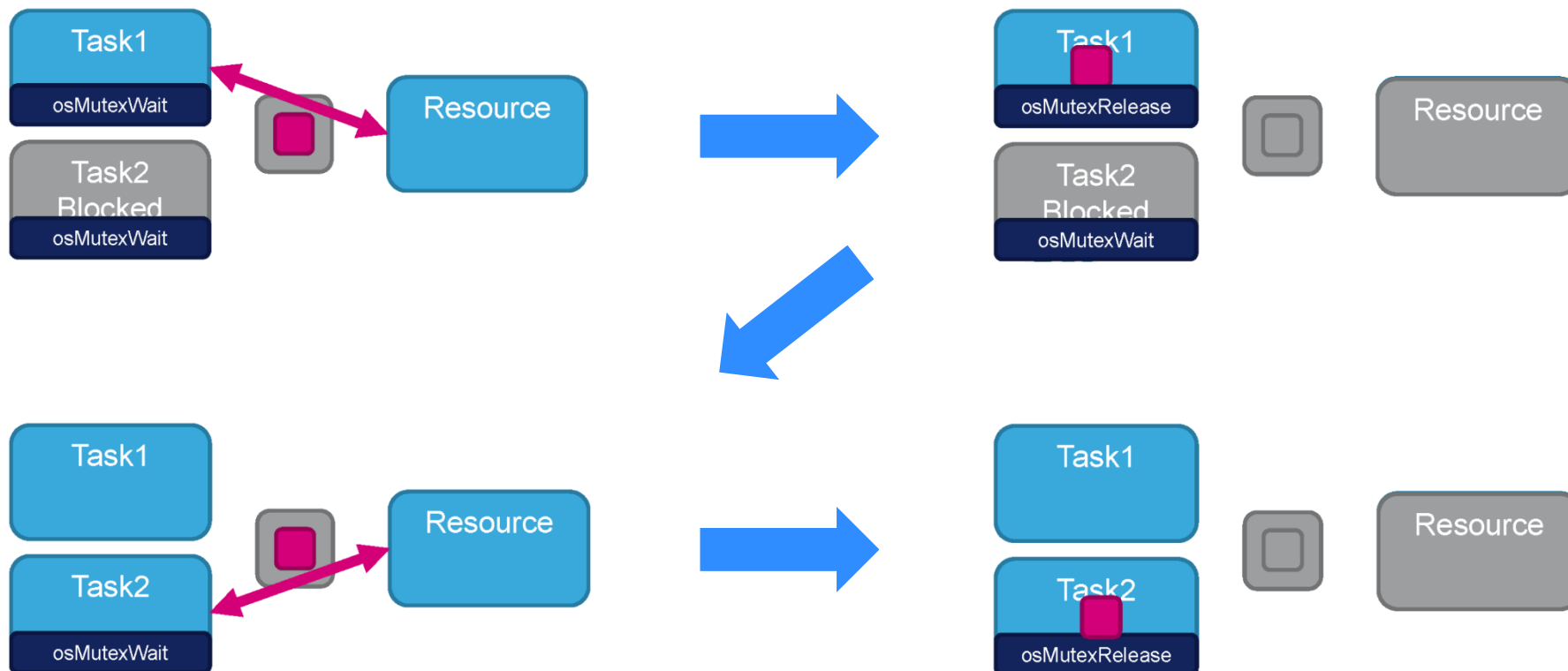
提交

任何两个并发进程之间存在着（ ）的关系。

- ☐ A 各自完全独立
- ☐ B 拥有共享变量
- ☐ C 必须互斥
- ☒ D 可能彼此制约

提交

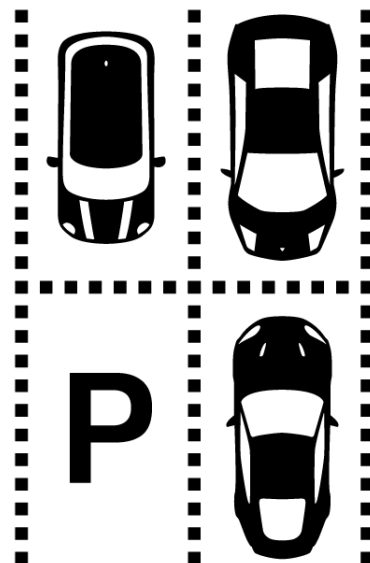
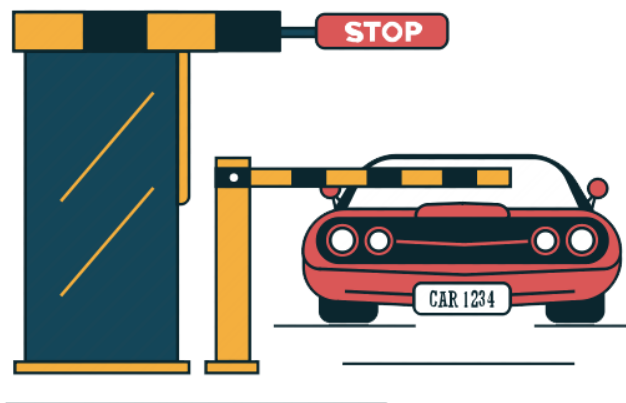
Mutex使用状态示例

**临界资源数量 = 1**

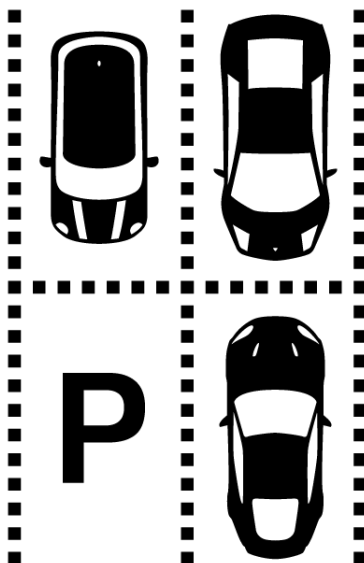
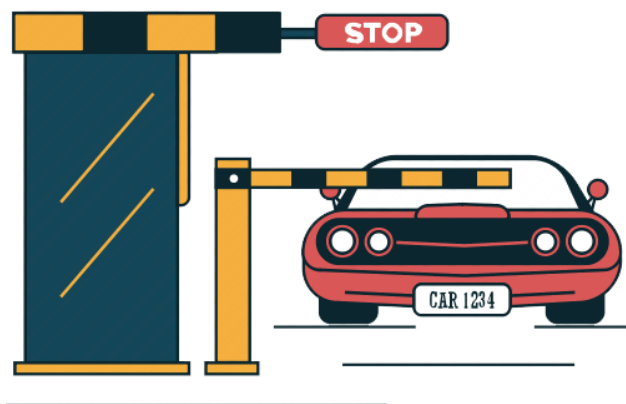
多资源实例的临界区管理

临界资源数量 = $n > 1$

典型示例: Car parking



Car Parking示例状态分析

**P**_{parking}

Check at the parking lot; 申请一个停车位

进入停车场, 占据一个停车位;

Exit the parking lot; 释放一个停车位

初始状态: 临界资源 (车位) 数 $S = n = 4$ 来一辆车c1: 进入, $S = 3$ 再来一辆车c2: 进入, $S = 2$ 再来一辆车c3: 进入, $S = 1$ 再来一辆车c4: 进入, $S = 0$

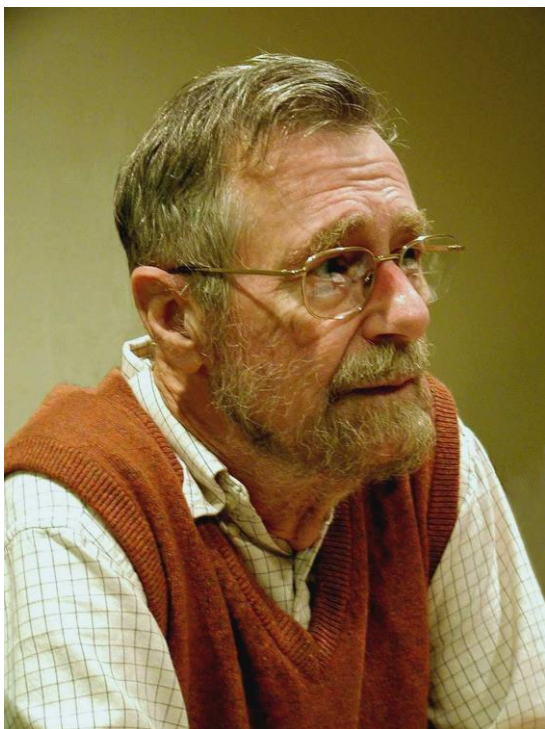
再来一辆车c5: 进不去了, 只能在入口等

信号量 (Semaphore) 的概念

定义：一个整型量 S ，对应两个原语操作 P 和 V

P : 申请1个资源（若资源量大于0，则该操作成功，资源量减一）

V : 释放1个资源（资源量增一）

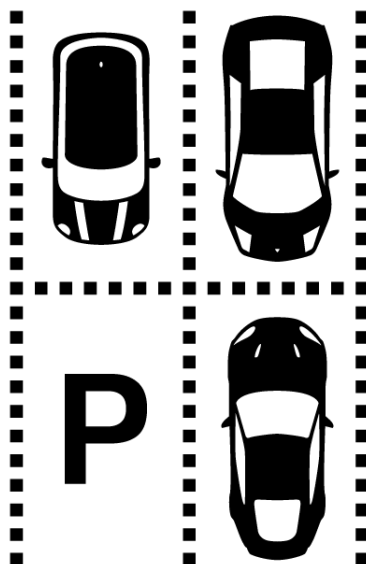
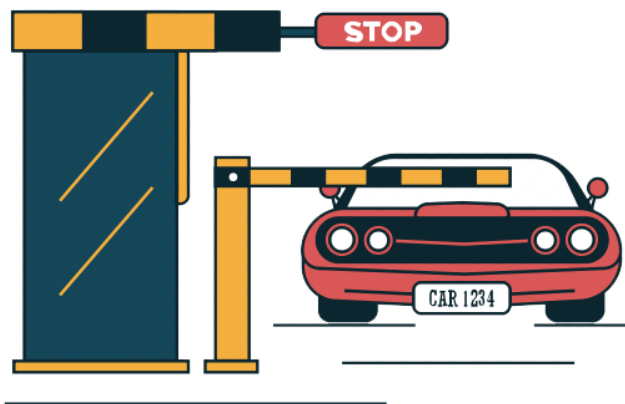


1965年，由荷兰科学家Dijkstra提出

P – proberen (测试, 荷兰语) / wait (英语)

V – verhogen (增量, 荷兰语) / signal (英语)

基于信号量解决Car Parking示例



P_{parking}

$S=4$

$P(S)$; 申请一个停车位

进入停车场, 占据一个停车位;

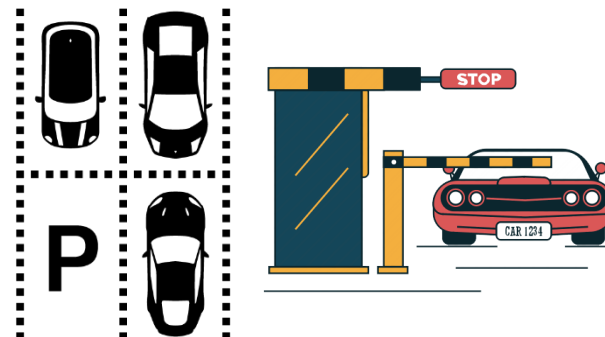
$V(S)$; 释放一个停车位

基于忙等方式实现信号量

S 初值=资源数的初值

```
Wait(S){ //P操作
    while(S <= 0);
    S--;
}
```

```
Signal(S){ // V操作
    S++;
}
```



试分析：在这种实现下，对于Car Parking问题的处理效果

P_{parking}

$S=4$

$P(S)$; 申请一个停车位

进入停车场，占据一个停车位；

$V(S)$; 释放一个停车位

基于忙等的信号量实现的问题分析

信号量值S

```
Wait(){ //P操作
    while(S<=0);
    S--;
}
```

```
Signal(){// V操作
    S++;
}
```

问题:


- ① 忙等 = 低效
- ② 无法记录等待资源的进程数量

如何避免这个问题的呢?

利用进程的阻塞状态, 避免忙等

方案: 当一个进程在利用wait () 操作进行资源申请时, 若发现所需资源已经被分配完毕时, 必须实现等待效果, 即此时让执行wait()操作的进程进入阻塞状态

对信号量数据结构作针对性修改:



```
typedef struct {
    int value;
    struct process *L;
}semaphore;
```

← 在该信号量上阻塞的进程队列



改进的信号量实现

```
void wait(semaphore S){
    S.value--;
    if(S.value<0){
        add this process to S.L;
        block();
    }
}
```

```
void signal(semaphore S){
    S.value++;
    if(S.value<=0){
        remove a process P from S.L;
        wakeup(P);
    }
}
```

忙等问题是否已解决?

等待资源的进程数量是否得到记录?



1.处理临界区问题

信号量mutex=1

```
do{  
    wait(mutex);  
    临界区;  
    signal(mutex);  
    退出区;  
}while(1);
```

2.处理多资源实例的资源竞争

信号量seat=10

```
//有10个座位的自习室管理  
do{  
    wait(seat);  
    进自习室自习;  
    signal(seat);  
}while(1);
```

完成对进程间接协作问题的统一处理

有一阅览室，读者进入时必须在一张表上进行登记，该表为每一个座位列出一个表目（包括座号、姓名、阅览时间），读者离开时要撤销登记信息，阅览室有100个座位。请用P、V操作描述读者进程间的同步。

作答



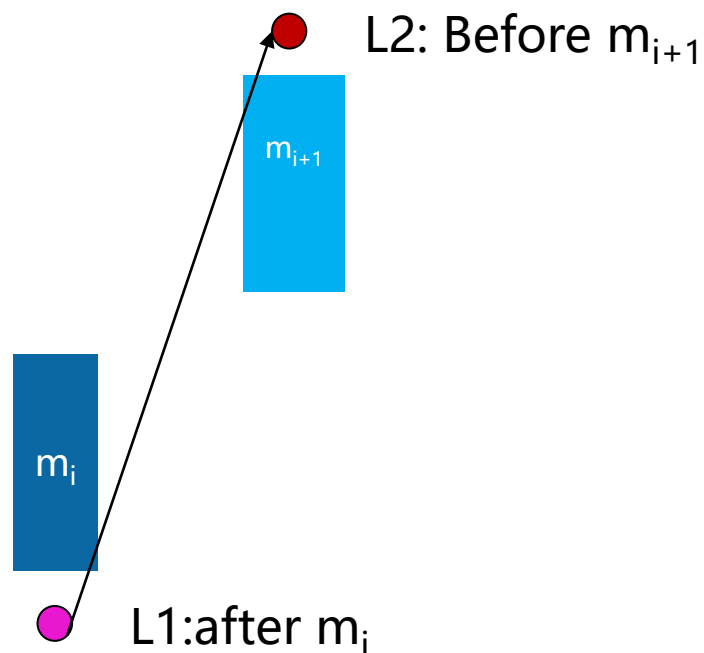
问题：我们做一个软件项目，分成几个部分 m_1, m_2, \dots, m_k
必须做完 m_i 之后，才能开始做 m_{i+1}

```
do{  
    实施 $m_i$   
    signal( $sem_i$ );  
}while(1);
```

```
do{  
    wait( $sem_i$ );  
    ...  
}while(1);
```

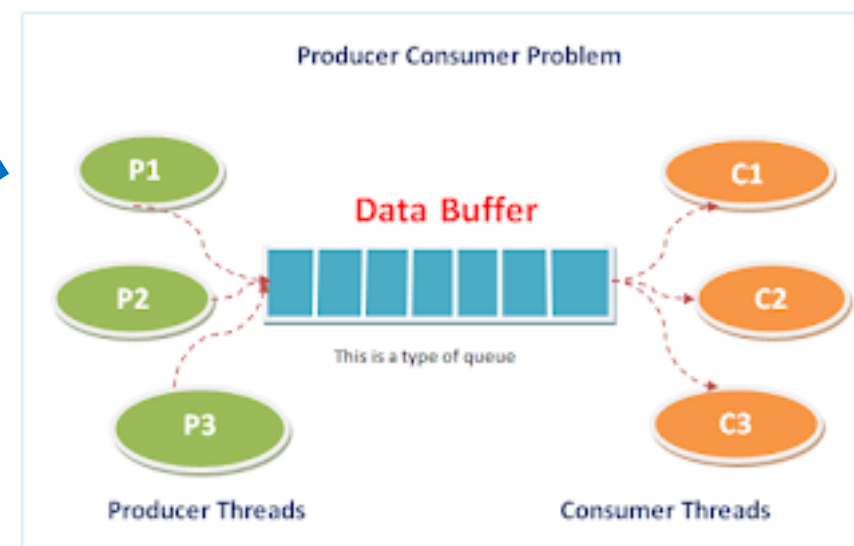
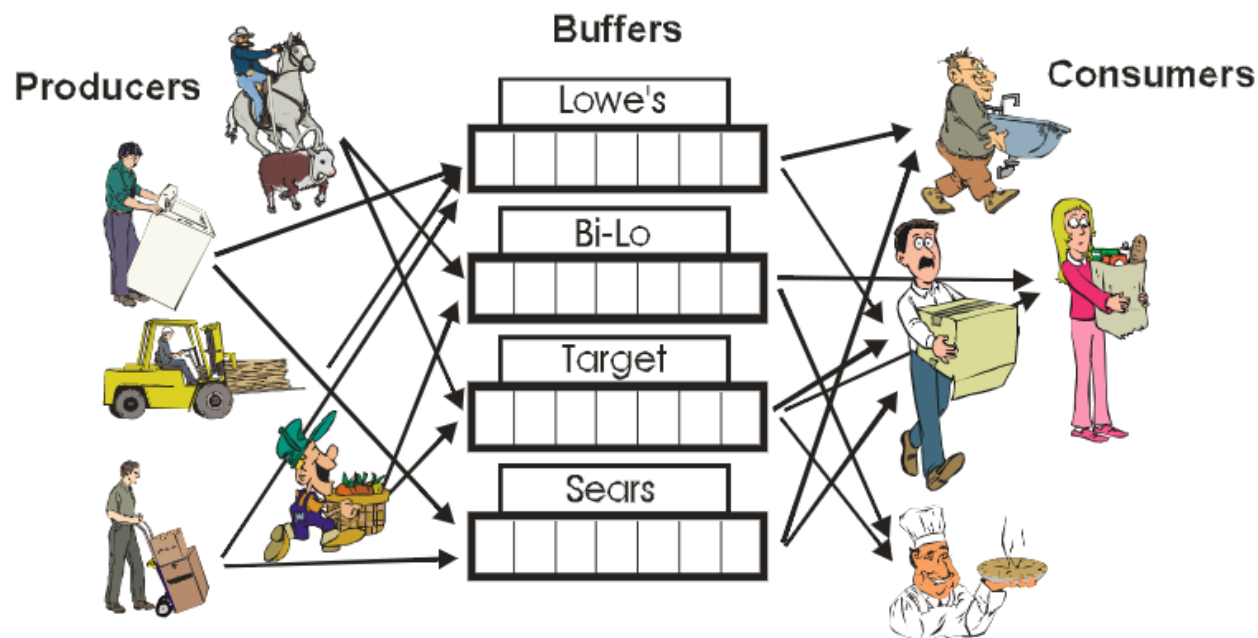
sem_i 的初值=?

完成对进程直接协作问题的统一处理



如果 P_{i+1} 先执行到L2，则得必须等待

采用初值为0的信号信号量，相当于先把从 m_i 到 m_{i+1} 的通路锁住，如果 P_{i+1} 本应后发的任务先至，则因为通道被锁，过不去等到先序进程 P_i 完成 m_i 后，将通道锁打开， P_{i+1} 才能继续



- Producers-**Stores**-Consumers
- 生产者消费者问题 (Producer Consumer Problem)
 - 又称 **Bounded Buffer Problem**
 - 给定有限大小的缓冲区，生产者将生产出的产品放入缓冲区，消费者从缓冲区取出产品

• 最简化的版本

- 1个生产者，1个消费者，共享缓冲区大小=1

1 Producer (生产者)

1 Consumer (消费者)

放入产品

取出产品



• 进程协作关系分析步骤

1. 理解问题本质，列出涉及的进程
2. 分析进程的协作关系
3. 根据进程协作关系设立信号量，并在程序合理位置通过P/V操作施加并发控制

• 步骤1: 列出问题所涉及的进程

```
P:
while (true) {
    生产一个产品;
    将产品放入缓冲区;
};
```

```
C:
while (true) {
    从缓冲区取产品;
    消费产品;
};
```

• 步骤2: 分析进程协作关系

- 2.1 进程P要执行put操作，要确定缓冲区内有空位存放产品，而除了初始状态之外，这种空位需要进程C的get操作（消费）来创造
- 2.2 进程C要执行get操作，要确定缓冲区内有存放至少1个产品，产品需要进程P通过生产并通过put操作放入缓冲区

• 步骤3: 设立信号量，通过P/V操作施加并发控制

- 信号量empty=1
- 信号量full=0
- 信号量的P/V操作应该加到进程合理的位置

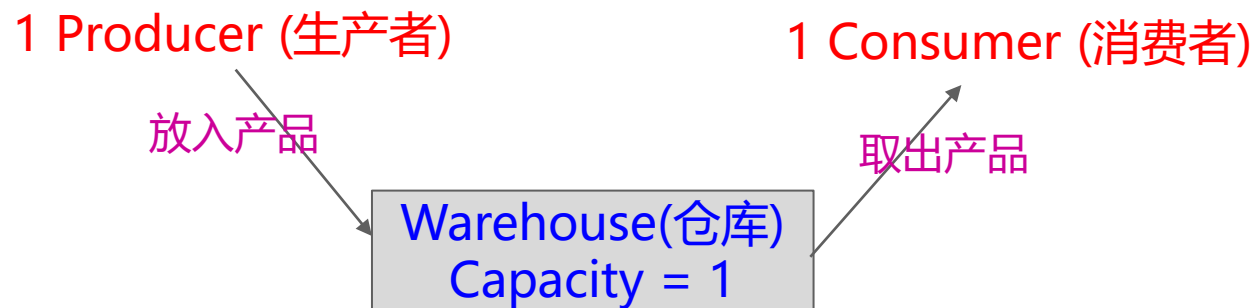


6.7-生产者消费者问题

P-C问题：最简形态

- 最简化的版本

- 1个生产者，1个消费者，共享缓冲区大小=1



- 进程协作关系分析步骤

1. 理解问题本质，列出涉及的进程
2. 分析进程的协作关系
3. 根据进程协作关系设置信号量，并在程序合理位置通过P/V操作施加并发控制

简版P-C问题 Solution:

```
int empty = 1;
```

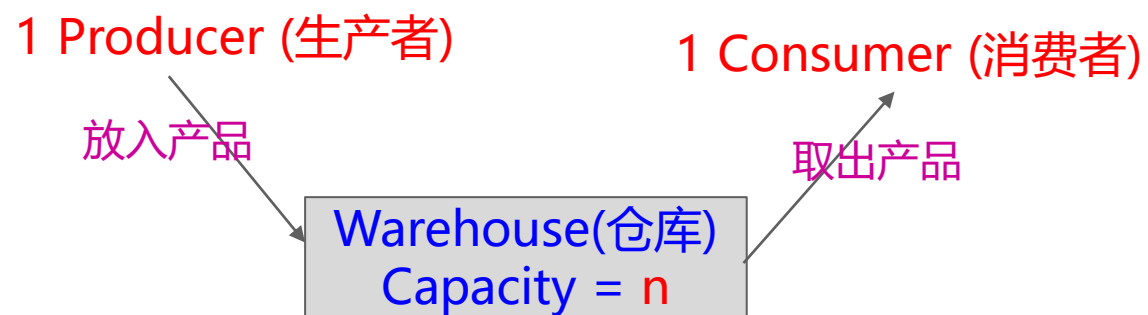
```
int full = 0;
```

```
P:
while (true) {
    生产一个产品;
    P(empty);
    送产品到缓冲区;
    V(full);
};
```

```
C:
while (true) {
    P(full);
    从缓冲区取产品;
    V(empty);
    消费产品;
};
```



- 扩展版：缓冲区大小 $1 \rightarrow n$





- 扩展版：缓冲区大小 $1 \rightarrow n$

Solution:

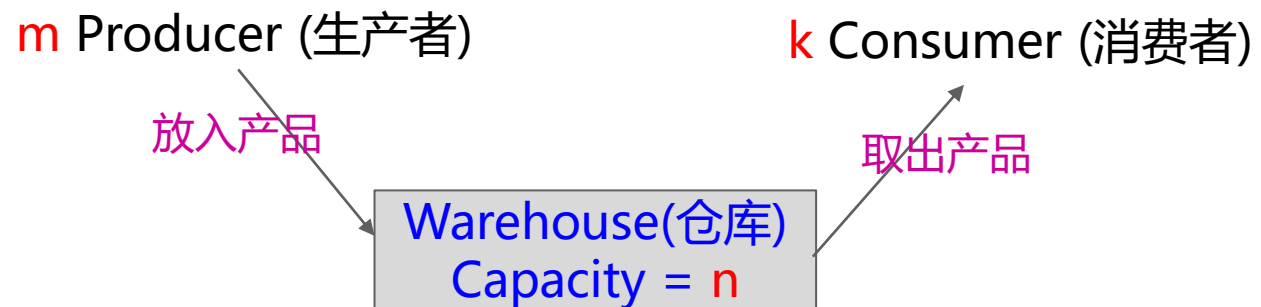
`int empty = n;`

```
P:
    i = 0;
    while (true) {
        生产产品;
        P(empty);
        往Buffer [i]放产品;
        i = (i+1) % n;
        V(full);
    };
```

`int full = 0;`

```
C:
    j = 0;
    while (true) {
        P(full);
        从Buffer[j]取产品;
        j = (j+1) % n;
        V(empty);
        消费产品;
    };
```

仓库大小为 n ($n > 1$) 时的同步控制





Solution:

```
mutex=1
```

```
int empty = n;
```

```
int full = 0;
```

```
P:
while (true) {
    生产产品;
    P(empty);
    P(mutex);
    往Buffer [i]放产品;
    i = (i+1) % n;
    V(mutex);
    V(full);
};
```

```
C:
while (true) {
    P(full);
    P(mutex);
    从Buffer[j]取产品;
    j = (j+1) % n;
    V(mutex);
    V(empty);
    消费产品;
};
```

小结:  信号量概念

 信号量用法

 经典同步问题1: 生产者消费者问题



有一只铁笼子，每次只能放入一只动物，猎手向笼子里放入老虎，农民向笼子里放入猪；动物园等待取笼子里的老虎，饭店等待取笼子里的猪。请对此问题进行进程同步分析，并用信号量机制给出解决问题的进程伪码。



某寺庙，有小和尚、老和尚若干。庙内有一水缸，由小和尚提水入缸，供老和尚饮用。水缸可容纳 30 桶水，每次入水、取水仅为1桶，不可同时进行。水取自同一井中，水井径窄，每次只能容纳一个水桶取水。设水桶个数为5个，试用信号灯和PV操作给出老和尚和小和尚的活动。



和尚取水问题: 分析进程间协作关系

```
young_monk(){
```

```
    while(1){
```

```
        go to the well;
```

```
        get water;
```

```
        go to the temple;
```

```
        pour the water into the big jar;
```

```
    }
```

```
}
```

```
old_monk(){
```

```
    while(){
```

```
        get water;
```

```
        drink water;
```

```
    }
```

```
}
```

分析其中的协作关系...



和尚取水问题: 分析资源, 设定信号量

```
young_monk(){
```

```
    while(1){
```

```
        go to the well;
```

```
        get water;
```

```
        go to the temple;
```

```
        pour the water into the big jar;
```

```
    }
```

```
}
```

```
old_monk(){
```

```
    while(){
```

```
        get water;
```

```
        drink water;
```

```
    }
```

```
}
```

semaphore empty=30; // 表示缸中目前还能装多少桶水, 初始时能装**30**桶水

semaphore full=0; // 表示缸中有多少桶水, 初始时缸中没有水

semaphore buckets=5; // 表示有多少只空桶可用, 初始时有**5**只桶可用

semaphore mutex_well=1; // 用于实现对井的互斥操作

semaphore mutex_bigjar=1; // 用于实现对缸的互斥操作



和尚取水问题: solution

young_monk(){

while(1){

P(empty);

P(buckets);

go to the well;

P(mutex_well);

get water;

V(mutex_well);

go to the temple;

P(mutex_bigjar);

pure the water into the big jar;

V(mutex_bigjar);

V(buckets);

V(full);

}

old_monk(){

while(){

P(full);

P(buckets);

P(mutex_bigjar);

get water;

V(mutex_bigjar);

drink water;

V(buckets);

V(empty);

}

}

}



一座小桥(最多只能承重两个人)横跨南北两岸, 任意时刻同一方向只允许一人过桥, 南侧桥段和北侧桥段较窄只能通过一人, 桥中央一处宽敞, 允许两个人通过或歇息。试用信号量和PV操作写出南、北两岸过桥的同步算法。



桥上可能没有人，也可能有一人，也可能有两人。

共需要四个信号量：

load1来控制桥上可向南人数，初值为1；

load2来控制桥上可向北人数，初值为1；

north用来控制北段桥的使用，初值为1，用于对北段桥互斥；

south用来控制南段桥的使用，初值为1，用于对南段桥互斥。



```
tosouth(){  
    过北段桥;  
    到桥中间;  
  
    过南段桥;  
    到达南岸  
}
```

```
tonorth(){  
    过南段桥;  
    到桥中间  
  
    过北段桥;  
    到达北岸  
}
```

```
semaphore load1=1,load 2 =1;  
semaphore north=1;  
semaphore south=1;
```



```
tosouth(){  
    P(load1);  
    P(north);  
    过北段桥;  
    到桥中间;  
    V(north);  
    P(south);  
    过南段桥;  
    到达南岸  
    V(south);  
    V(load1);  
}
```

```
tonorth(){  
    P(load2);  
    P(south);  
    过南段桥;  
    到桥中间  
    V(south);  
    P(north);  
    过北段桥;  
    到达北岸  
    V(north);  
    V(load2);  
}
```



有一座东西向的独木桥，用信号量P/V操作实现：

- (1) 每次只允许一个人过桥；
- (2) 当独木桥上有行人时，同方向的行人可以过桥，相反方向的人必须等待。



n 个并发进程，信号量初始值为 1，当 n 个进程都执行 P 操作后，信号量的值为()。



信号量初值为 4，多次 PV 操作后变为 -2，那么当前时刻已经顺利获得资源的进程数目 = ()。



5 个并发进程，信号量初始值为 3，那么信号量取值范围是整数区间为 $[(), ()]$ 。



谢谢!
Thank you!