Operating Systems

Chapter 5 Mutual Exclusion(互斥) and Synchronization(同步)

经典习题

review

- 一个生产者生产 N次

```
void produce () {
        while(true) {
                i++// 无需互斥
                semwait(mayprdc)
                outputs: “producing %d”,i
                semsignal(semg);
}
```

review

- N个生产者竞争生产，每个只生产一次
  - int i=0
  - 如果 mayprdc 只能是 0 和 1（起到互斥作用），那么将 i++ 放入 semwait 后面即可
  - 如果 mayprdc 可以 >1（起不到互斥作用），那么将 i++ 需要额外的互斥锁 mutex

```
void produce 1() {
        i++;// 需要互斥
        semwait(mayprdc)
        outputs: “producing %d”,i
        semsignal(mayconsume);
}
```

```
void produce n() {
        i++; // 需要互斥
        semwait(mayprdc)
        outputs: “producing %d”,i
        semsignal(mayconsume);
}
```

Agenda

- 1. 吃水果问题
- 2. 抽烟者问题
- 3. 过桥问题
- 4. 理发师问题
- 5. 超市问题
- 6. 数据搬移
- 7. 快递柜问题

1. 吃水果

- 吃水果问题：桌子有一只盘子，只允许放一个水果，父亲专向盘子放苹果，母亲专向盘子放桔子儿子专等吃盘子的桔子，女儿专等吃盘子的苹果。只要盘子为空，父亲或母亲就可以向盘子放水果， 仅当盘子有自己需要的水果时，儿子和女儿可从盘子取出。

1. 吃水果

- 生产者：父母
  - 等待信号：盘子空
  - 发出信号：某个水果生产完成
- 消费者：儿女
  - 等待信号：某个水果生产完成
  - 发出信号：盘子空
- 互斥数据
  - 盘子

1. 吃水果

apple=0,orange=0,empty=1(对应盘子互斥)

```
void father() {
        while(true) {
                semwait(empty)
                output ： apple ready…
                semsignal(apple)
        } }
```

```
void son(){
    while(true) {
                semwait(orange);
                output ： eat orange …
                semsignal(empty)
}}
```

```
void mother() {
        while(true) {
                semwait(empty)
                output ： orange ready…
                semsignal(orange )
}}
```

```
void daughter(){
    while(true) {
                semwait(apple);
                output ： eat apple…
                semsignal(empty)
}}
```

2. 抽烟者问题

- 抽烟者问题。假设一个系统中有三个抽烟者进程，每个抽烟者不断地卷烟并抽烟。抽烟者卷起并抽掉一颗烟需要有三种材料：烟草、纸和胶水。一个抽烟者有烟草，一个有纸，另一个有胶水。系统中还有两个供应者进程，它们无限地供应所有三种材料，但每次仅轮流提供三种材料中的两种。得到缺失的两种材料的抽烟者在卷起并抽掉一颗烟后会发信号通知供应者，让它继续提供另外的两种材料。

2. 抽烟者问题

- 抽烟者 tobacco(烟草)paper(纸)glue(胶水)
  - 等待各自信号 semt,semp,semg, 等到则集齐材料 (同步)
  - 发出下一个可以生产信号 mayprdc
- 生产者 produce1, produce2
  - 等待可以生产信号 (互斥) mayprdc
  - 发出 semt,semp,semg 之一 (同步及互斥)

2. 抽烟者问题

semt=0,semp=0,semg=0,mayprdc=1

```
void produce1() {
        while(true) {
                semwait(mayprdc)
                num=random%3;
                if(num==0)
                    semsignal(semt);
                else if(num==1)
                    semsignal(semp);
                else
                    semsignal(semg);
}
```

```
void produce2() {
        while(true) {
                semwait(mayprdc)
                num=random%3;
                if(num==0)
                    semsignal(semt);
                else if(num==1)
                    semsignal(semp);
                else
                    semsignal(semg);
}
```

2. 抽烟者问题

semt=0,semp=0,semg=0,mayprdc=1

```
void tobacco() {
        while(true) {
                semwait(semt)
                 output: t is smoking
                semsignal(mayprdc);
}
```

```
void glue() {
        while(true) {
                semwait(semg)
                 output: g is smoking
                semsignal(mayprdc);
}
```

```
void paper() {
        while(true) {
                semwait(semp)
                 output: p is smoking
                semsignal(mayprdc);
}
```

3. 过桥问题

- 有一座桥，东西走向，汽车可以从东往西走，也可以西往东走，桥上每次只允许朝一个方向走，请用信号量描述下列过程：
  - 如果某一个方向的车占有桥，让这个方向的车优先过桥；

3. 过桥问题

1. 问题 1 的模型和读者写者问题中的读者优先问题类似，在问题中存在两类进程，从东往西走的车，和从西往东走的车，两者存在竞争关系，竞争争夺桥的使用权，这是互斥关系，同时对于两类进程之间有存在两个竞争关系，对进程的数目进行统计。所以这个问题是一个典型的互斥关系；

1. 初始化：全局变量 ecount 和 wcount 分别统计两边过桥的汽车数目
2. 定义三个信号量： mutex=1(桥的互斥),emutex=1(对 ecount 修改的互斥),wmutex=1(对 wcount 修改的互斥);

3. 过桥问题

```
ecount =0,wcount=0,emutex=1,wmutex=1,mutex=1
```

```
void  e2w(){
    semwait(emutex)
    ecount++
    if (ecount==1)
            semwait(mutex)
    semsignal(emutex)
    cross the bridge from east to west
    semwait(emutex)
    ecount--
    if (ecount==0)
            semsignal(mutex)
    semsignal(emutex)
}
```

```
void  w2e(){
    semwait(wmutex)
    wcount++
    if (wcount==1)
            semwait(mutex)
    semsignal(wmutex)
    cross the bridge from west to east
    semwait(wmutex)
    wcount--
    if (wcount==0)
            semsignal(mutex)
    semsignal(wmutex)
}
```

3. 过桥问题

- 修改方案：让两个方向的车公平的过桥。

- 隐藏涵义：东西方向的车在同一个队列（ 信号量）上排队
- 信号量 queue=1, 线程都需要先 semwait(queue)
  计数信号量

3. 过桥问题

```
void  e2w(){
   semwait(queue);
   semwait(emutex)
    ecount++
    if (ecount==1)
            semwait(mutex)
    semsignal(emutex)
    cross the bridge from east to west
    semwait(emutex)
    ecount--
    if (ecount==0)
            semsignal(mutex)
    semsignal(emutex)
}
```

```
void  w2e(){
     semwait(queue);
     semwait(wmutex)
     wcount++
     if (wcount==1)
            semwait(mutex)
      semsignal(wmutex)
      cross the bridge from west to east
       semwait(wmutex)
       wcount--
     if (wcount==0)
            semsignal(mutex)
     semsignal(wmutex)
}
```

3. 过桥问题

- 同时要满足如果 east 方向的车正在通行，又来一辆 east 方向的车，而第三辆是 west 方向的车，那么第二辆车可以在第一辆通行过程中加入（依然是读者问题）
- Semsignal(queue) 的位置？

- 按照读者问题的解决办法，加在读者排队 semwait(emutex) 和 semwait(wmutex) 后面

3. 过桥问题

```
void  e2w(){
   semwait(queue);
   semwait(emutex)
    ecount++
    if (ecount==1)
            semwait(mutex)
    semsignal(queue)
    semsignal(emutex)
   // semsignal(queue)
    cross the bridge from east to west
    semwait(emutex)
    ecount--
    if (ecount==0)
            semsignal(mutex)
    semsignal(emutex)
}
```

```
void  w2e(){
     semwait(queue);
     semwait(wmutex)
     wcount++
     if (wcount==1)
            semwait(mutex)
      semsignal(queue)
     semsignal(wmutex)
     //semsignal(queue)
     cross the bridge from west to east
      semwait(wmutex)
      wcount--
     if (wcount==0)
            semsignal(mutex)
     semsignal(wmutex)
}
```

4. 理发师问题

- 一个理发店有 1 个理发师，n 张椅子，1 张理发椅。若没有要理发的顾客，则理发师就去睡觉；若有顾客走进理发店且所有的椅子都被占用了，则该顾客就离开理发店；若理发师正在为人理发，则该顾客就找一张空椅子坐下等待；若理发师在睡觉，则顾客就唤醒他，请用信号量描述这个过程。

4. 理发师问题

- 理发师：睡觉 / 工作
  - 关键：收到“有客人“信号就工作（同步问题）
  - 工作内容：输出消息，凳子数 +1（互斥）
- 客人：理发 + 等待 / 离开
  - 关键：位置
    - 有位置（互斥）则等待，凳子数 -1（互斥），通知理发师（同步）
    - 无位置（互斥）离开
- 互斥数据：凳子数

4. 理发师问题

```
void baber()
{
   while(true) {
          semwait(guest)# 唤醒
          semwait(mutex)
          count--;
          semsignal(mutex)
      }
}
```

```
void customer(){
    while(true) {
          semwait(mutex);
        if (count<n+1) {
            count++
            if(count==1)
                semsignal(guest)
            else
        set on chair waiting…
        }
        else{
            leave the baber's
        }
            semsignal(mutex)

}
```

4. 理发师问题

```
void baber()
{
   while(true) {
          semwait(guest)# 唤醒
          semwait(mutex)
          count--;
          if(count!=0)
                semsignal(guest)// 别睡
          semsignal(mutex)
      }
}
```

```
void customer(){
    while(true) {
          semwait(mutex);
        if (count<n+1) {
            count++
            if(count==1)
                  semsignal(guest)
            else
        set on chair waiting…
        }
        else{
            leave the baber's
        }
            semsignal(mutex)

}
```

1. 理发师问题

```
void baber()
{
   while(true) {
          semwait(guest)# 唤醒
          semwait(mutex)
          count--;
          output ： serving…
          semsignal(mutex)
      }
}
```

```
void customer(){
    while(true) {
          semwait(mutex);
        if (count<n+1) {
            count++
            semsignal(guest)
             output ： waiting…
            semsignal(mutex)
        }
        else{
            output ： leave the baber's
            semsignal(mutex)
        }
}
```

4. 理发师问题

- 测试需要观察：等待，理发，不等待离开 交替执行
  - 1. 理发师线程先激活
  - 2. 客户线程先激活
  - 3. 两个线程交叠执行

5. 超市购物问题

超市可以容纳 500 人同时购物，有 6 扇可供出入的门，既可以进又可以出，每扇门只允许一个人通过，使用信号量解决一下问题：

a)描述购物过程；

b)如果增加一个限制条件：顾客进出必须走同一个门，这个过程又是怎么样的；

5. 超市购物问题

问题定性：一个混合问题（同步和互斥）。

同步：超市里面维持 500 个顾客的规模；

互斥：每道门都是临界资源；

进程的类别：两类代表进程，进入超市，离开超市。

初始化：

同步信号量一个，定义为 s=500,

互斥量 6 个，定义为 s1=s2=s3=s4=s5=s6=1

5. 超市购物问题

```
void enter()
{
        semwait(s);
        choose the door i
        semwait(si);
        cross the door  i;
        semsignal(si)
        buy something;

}
```

```
void leave()
{
        choose door i;
        semwait(si);
        cross the door;
        semsignal(si)
        semsignal(s)

}
```

5. 超市购物问题

```
void customer()
{
        semwait(s);
        choose random door i
        semwait(si);
        cross the door  i;
        semsignal(si)
        buy something;//sleep(random time)
                choose  random door i;
                semwait(si);
                cross the door;
                semsignal(si);
                semsignal(s);
}
```

5. 超市购物问题

如果增加一个限制条件：顾客进出必须走同一个门，这个过程又是怎么样的
；

```
void customer(){
        semwait(s);
        choose random door i
        semwait(si);
        cross the door  i;
        semsignal(si)
        buy something;//sleep(random time)
                semwait(si);
                cross the door;
                semsignal(si);
                semsignal(s);
}
```

6. 数据搬移

- 有三个进程： Read 、 Move 和 Print ，共享两个缓存 B1 和 B2
  - 进程 Read ：读取一条记录，并放在缓存 B1 中
  - 进程 Move ：从缓存 B1 中读取记录，处理后放入缓存 B2 中
  - 进程 Print ：从 B2 中读取数据并打印

  请通过信号量的等待和激发操作填空

6. 数据搬移

```
Initialize
Semaphore S0 = 1;
Semaphore S1 = 0;
Semaphore S2 = _____;(1)
Semaphore S3 = _____;(2)
```

Read Process	Move Process	Print Process

char x;
while (true)
{
Read a record to x;
_____;(3)
B1 = x;
_____;(4)
}
	char x, y;
	while (true)
	{
	_____;(5)
	x = B1;
	_____;(6)
	Process x,
	store the result to y;
	_____;(7)
	B2 = y;
	_____;(8)
	}
		char x;
		while (true)
		{
		_____;(9)
		x = B2;
		_____;(10)
		Print x;
		}

6. 数据搬移

- 知识点：互斥和同步、信号量
- Page 154 ，结合 book5.3 信号量
- 需要注意的地方：系统运行时，必须保证 READ 优先使用 B1 缓存， MOVE 优先使用 B2 缓存（原因：初始状态 B1 和 B2 缓冲中的数据是无效数据）

Problems 2: – 思路

- 三个进程的关系如下：
  - ✓ 对于 READ 进程：在对 B1 缓冲区读取数据时，首先要判断 MOVE 进程是否将上一次读的数据取走，如果没有取走， READ 等待；否则，读取一段数据放到 B1 ，然后通知 MOVE 进程， B1 缓存可用；
  - ✓ 对于 MOVE 进程：首先判断 B1 缓冲区中的数据是否可用，如果没有等待；否则，从 B1 缓存中读取数据，然后对数据进行加工，加工完毕后，判断 PRINT 进程是否已将上次 MOVE 进程放到 B2 缓冲区中的数据取走，如果没有取走，等待；否则，将加工后的数据存放到 B2 缓冲区，然后通知 PRINT 进程可以使用 B2 缓冲区；
  - ✓ 对于 PRINT 进程：判断 B2 缓冲区的数据是否可用，如果不可用，则等待；如果可用，则打印，然后通知 MOVE 进程， B2 的数据已取走， MOVE 进程可对 B2 缓冲区 进行更新操作。

6. 数据搬移

通过上面分析，可以知道 ，在这个程序中我们需要使用四个信号量，其中：

- S1 用于表明 READ 是否可以使用 B1 ；（ B1 中是否有空位置 ）
- S2 用于表明 MOVE 是否可以使用 B1 ；（ B1 中是否有数据 ）
- S3 用于表明 MOVE 是否可以使用 B2 ；（ B2 中是否有有空位置 ）
- S4 用于表明 PRINT 是否可以使用 B2 ；（ B2 中是否有数据 ）

6. 数据搬移

- 初始化：因为由于优先级的问题，所以有
  - Semsignal s1=1; （ B1 中有位置 ）1 表示有位置，0 表示没有位置
  - Semsignal s2=0; （ B1 中没有数据 ）1 表示有数据，0 表示没有数据
  - Semsignal s3=1; （ B2 中有位置 ）1 表示有位置，0 表示没有位置
  - Semsignal s4=0; （ B2 中没有 产品 ）1 表示有数据，0 表示没有数据

6. 数据搬移

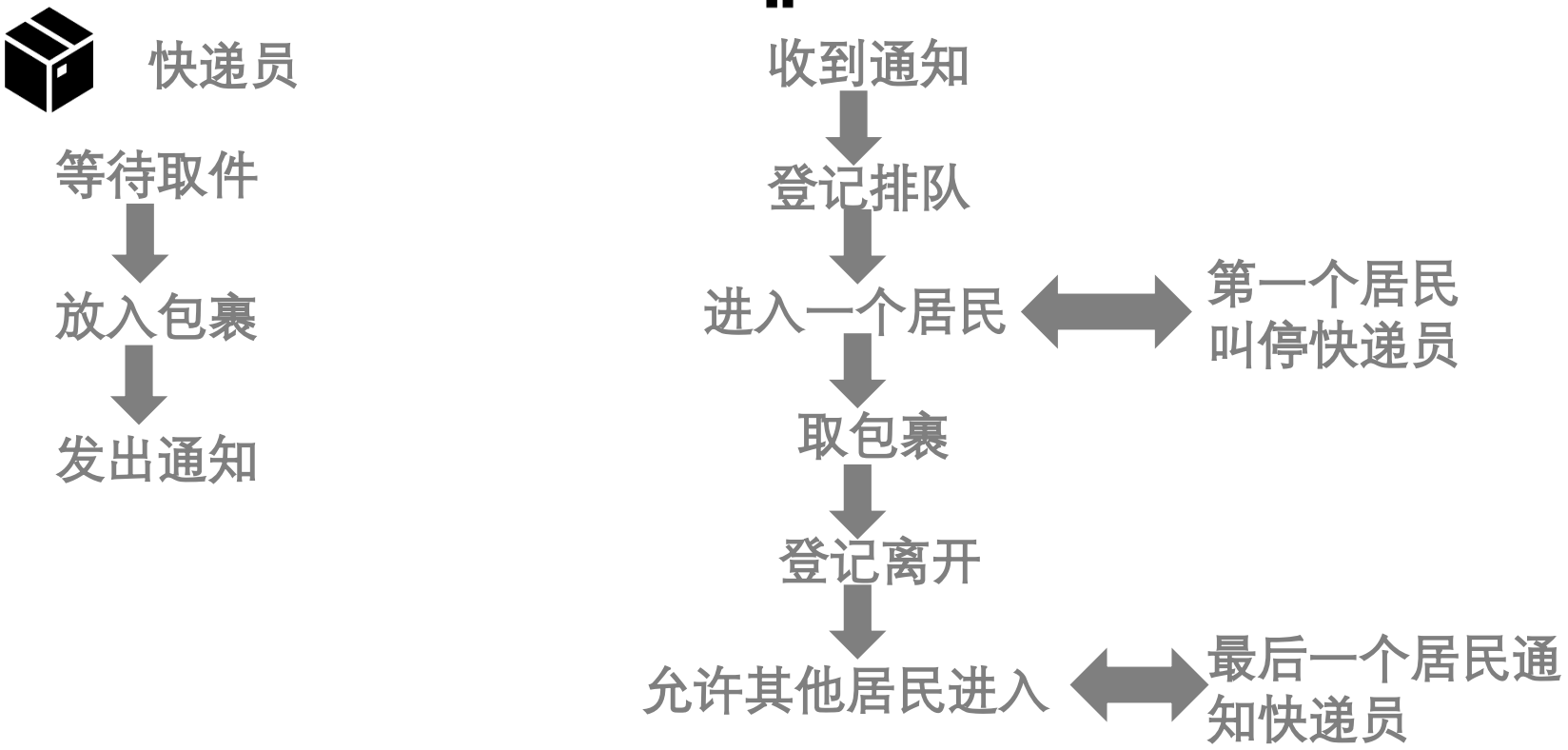
Read Process	Move Process	Print Process

char x;
while (true)
{
Read a record to x;
Semwait(s1);(3)
B1 = x;
Semsignal(s2);(4)
}
	char x, y;
	while (true)
	{
	Semwait(s2);(5)
	x = B1;
	semsignal(s1);(6)
	Process x,
	store the result to y;
	semwait(s3);(7)
	B2 = y;
	Semsignal(s4);(8)
	}
		char x;
		while (true)
		{
		Semwait(s4);(9)
		x = B2;
		semsignal(s3);(10)
		Print x;
		}

7. 快递柜问题

小区里有1个快递柜，该柜子有20个格子，快递员负责向快递柜放入包裹（每次只能放入1个包裹），放好一个包裹发出一个取件通知，若快递柜满了，则快递员需要等待空闲格子出来；居民凭快递通知，从指定快递柜中取走属于自己的包裹，每次只能允许一个居民取包裹；一个快递柜若有新的居民取包裹，快递员需要让居民先取包裹。假设初始时快递柜是空的，定义信号量并初始化，使用 P、V 操作模拟快递员和居民进程之间的同步与互斥。

7. 快递柜问题


快递员
等待取件
放入包裹
发出通知
居民
收到通知
登记排队
进入一个居民
第一个居民
叫停快递员
取包裹
登记离开
允许其他居民进入
最后一个居民通
知快递员

sem writemutex=1,readmutex=1**, z=1,mutex1=1,mutex2=1**;

```
void* courier (void *arg){
  while(1){

      semWait(writemutex)
      writedata();
      semSignal(writemutex)

  }}
```

```
void* resident(void *arg){
  while(1){

    semWait(mutex1)
    rcount++
    if(rcount==1)
        semWait(writemutex)
    semSignal(mutex1)

      readdata();

    semWait(mutex1)
    rcount--
    if(rcount == 0)
        semSignal(writemutex)
    semSignal(mutex1)

  }}
```

sem writemutex=1,readmutex=1**, z=1,mutex1=1,mutex2=1**;
Number=20

```
void* courier (void *arg){
  while(1){

      semWait(number)
     semWait(writemutex)
     writedata();
     semSignal(writemutex)

  }}
```

```
void* resident(void *arg){
  while(1){

   semWait(mutex1)
  rcount++
  if(rcount==1)
        semWait(writemutex)
   semSignal(mutex1)

    readdata();
   semsignal(number)

   semWait(mutex1)
   rcount--
    if(rcount == 0)
        semSignal(writemutex)
    semSignal(mutex1)

  }}
```

sem writemutex=1,readmutex=1**, z=1,mutex1=1,mutex2=1**;

Number=20

```
void* courier (void *arg){
  while(1){

      semWait(number)
     semWait(writemutex)
      writedata();
     semSignal(writemutex)

    semSignal(message i)//20 个消息

  }}
```

```
void* resident(void *arg){
  while(1){
 semwaitl(message i)
  semWait(mutex1)
 rcount++
 if(rcount==1)
        semWait(writemutex)
   semSignal(mutex1)
   semWait(mutextakeout)
    readdata();
   semSignal(mutextakeout)
  semsignal(number)
  semWait(mutex1)
  rcount--
   if(rcount == 0)
        semSignal(writemutex)
    semSignal(mutex1)
  }}
```

sem writemutex=1,readmutex=1**, z=1,mutex1=1,mutex2=1**;

Number=20 ， courier_queue=1

Mutextakeout=1

```
void* courier(void *arg){
  while(1){
      semWait(courier_queue)
      semWait(number)// 格子 i
      semWait(writemutex)// 储物柜
      writedata();
      semSignal(writemutex)
      semSignal(courier_queue)
      semSignal(message i)
  }}
```

```
void* resident(void *arg){
  while(1){
    semwaitl(message i)
    semWait(mutex1)
    rcount++
    if(rcount==1)
        semWait(writemutex)
    semSignal(mutex1)
    semWait(mutextakeout) // 储物柜
    readdata();
    semSignal(mutextakeout)
    semsignal(number)
    semWait(mutex1)
    rcount--
    if(rcount == 0)
        semSignal(writemutex)
    semSignal(mutex1)
  }}
```