第二章

并发进程

主 要 内 容

2.1 进程基本概念与调度控制

2.2 UNIX的进程

2.3 中断的基本概念及UNIX中断处理

2.4 进程通信与死锁

2.5 经典的进程通信问题

进程互斥

互斥：任意时刻只能有一个进程使用该资源

```
Process IN:
...
count++;
...
```

```
Process OUT:
...
count--;
...
```

```
mov ax, count;
inc ax;
mov count ax;
```

```
mov ax, count;
dec ax;
mov count ax;
```

互斥信号量：
semaphore mutex;
mutex.value = 1;

1．每一组相关临界区与一个信号量对应

2. 进入临界区：P操作
离开临界区：V操作

```
Process IN:
...
P(mutex);
count++;
V(mutex);
...
```

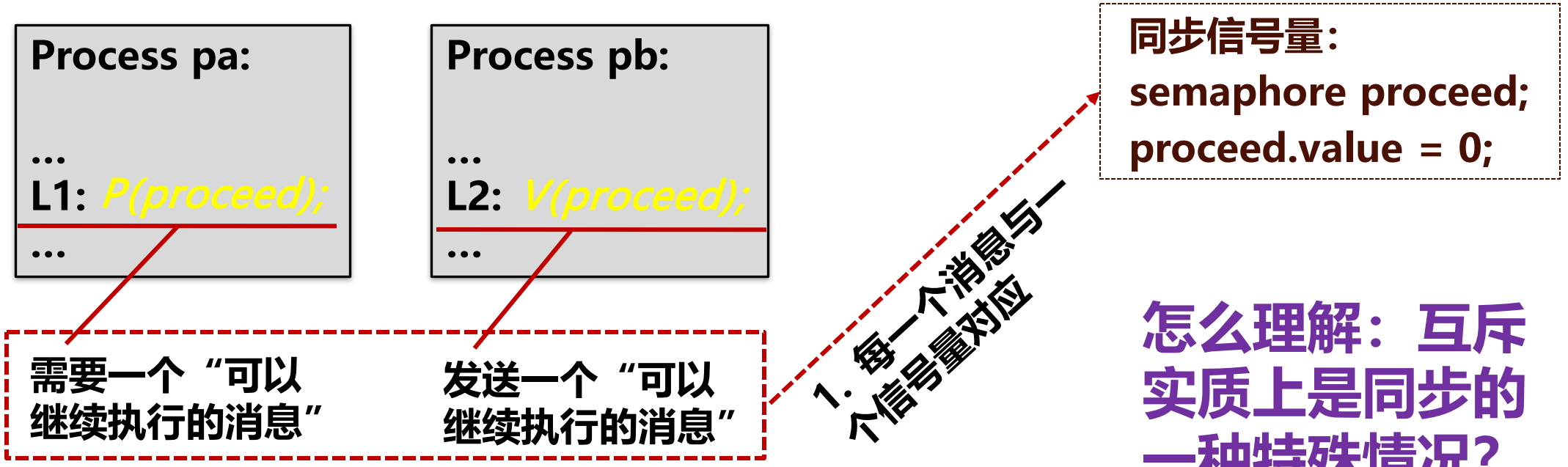
```
Process OUT:
...
P(mutex);
count--;
V(mutex);
...
```

3．PV操作在每个相关临界区前后对同一信号量成对使用

进程同步

利用信号量实现进程同步

同步：在同步点上等待“可以继续执行”的消息



怎么理解：互斥实质上是同步的一种特殊情况？

2. P操作：接收消息
   V操作：发送消息

3. PV操作由不同的进程实施，成对使用

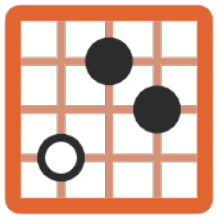
同步：直接的相互制约关系　　*互斥：间接的相互制约关系*

经典的进程通信问题

生产者 - 消费者问题

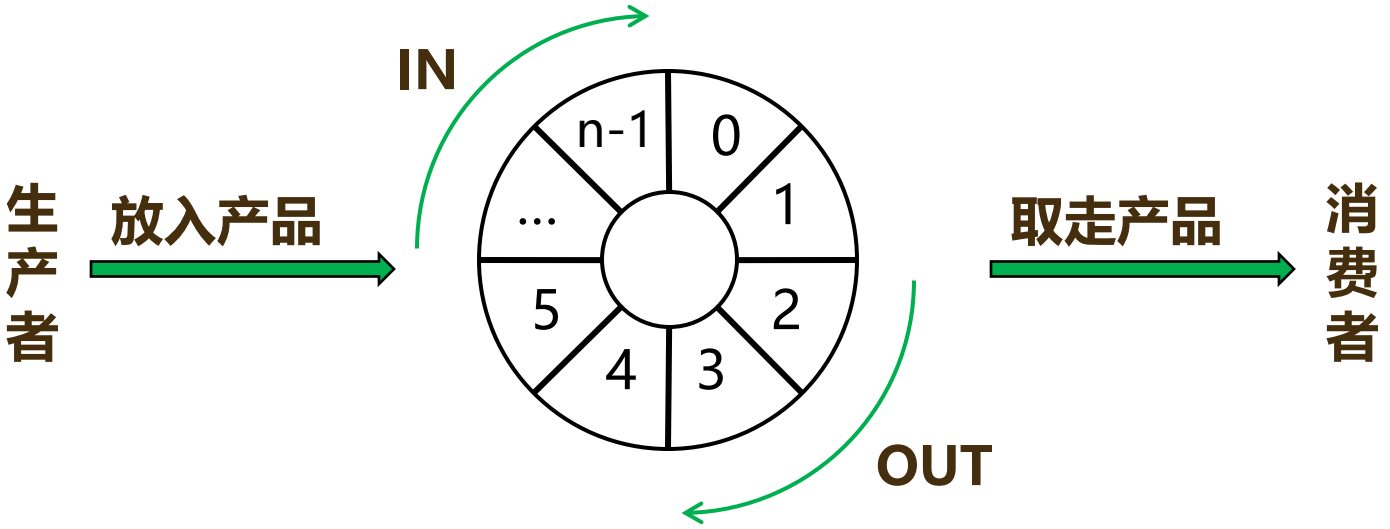
嗜睡理发师问题

读者 – 写者问题



黑白棋问题

1. 生产者——消费者问题



1. **生产者进程和消费者进程通过缓冲存储区发生联系。**
2. **生产者进程不断地执行“生产一个产品，将其放入缓冲区”的循环；消费者进程不断执行“从缓冲区取出一个产品，消耗使用该产品”的循环。**

1. 生产者——消费者问题

```
begin
    b: array[0..n-1] of integer;  k, t : integer;  k:=0;   t:=0;

cobegin:
PROCESS producer                    PROCESS consumer
begin                               begin
11:       produce item;             12:

                                              item = b[t];
          b[k]:=item;                         t:=(t+1) mod n;
          k:=(k+1) mod n;

                                              consume item;
          goto 11;                            goto 12;
end;                                end;
coend;
end;
```

1. 生产者——消费者问题

```
begin
    b: array[0..n-1] of integer;  k, t : integer;  k:=0;   t:=0;

cobegin:
PROCESS producer                    PROCESS consumer
begin                               begin
11:       produce item;             12:
          p(empty);
                                                item = b[t];
          b[k]:=item;                           t:=(t+1) mod n;
          k:=(k+1) mod n;
                                                v(empty);
                                                consume item;
          goto 11;                              goto 12;
end;                                end;
coend;
end;
```

1. 有空缓存单元时，生产者才能放入产品，若缓存区满，需等待消费者取走产品

生产者在放入一件产品前，
需要“有一个空单元”的消息
消费者在取走一个产品后，
发送“有一个空单元”的消息

empty : semaphore;

empty.value : = n ;

1. 生产者——消费者问题

```
begin
    b: array[0..n-1] of integer;  k, t : integer;  k:=0;   t:=0;

cobegin:
PROCESS producer                    PROCESS consumer
begin                               begin
11:     produce item;               12:     p(full);
        p(empty);
                                            item = b[t];
        b[k]:=item;                         t:=(t+1) mod n;
        k:=(k+1) mod n;
                                            v(empty);
        v(full);                            consume item;
        goto 11;                            goto 12;
end;                                end;
coend;
end;
```

2. 有满缓存单元时，消费者才能取走产品，若缓冲区空，需等待生产者放入产品

消费者在取走一件产品前，
需要“有一个满单元”的消息
生产者在放入一个产品后，
发送“有一个满单元”的消息

full : semaphore;

full.value : = 0 ;

1. 生产者——消费者问题

```
begin
    b: array[0..n-1] of integer;  k, t : integer;  k:=0;   t:=0;

cobegin:
PROCESS producer                    PROCESS consumer
begin                               begin
11:     produce item;               12:     p(full);
        p(empty);                           p(mutex);
        p(mutex);                           item = b[t];
        b[k]:=item;                         t:=(t+1) mod n;
        k:=(k+1) mod n;                     v(mutex);
        v(mutex);                           v(empty);
        v(full);                            consume item;
        goto 11;                            goto 12;
end;                                end;
coend;
end;
```

3. 缓冲区为共享存储区，生产者和消费者不能同时访问

mutex : semaphore;

mutex.value : = 1 ;

互斥信号量和同步信号量的位置调换？

1. 生产者——消费者问题

```
begin
    b: array[0..n-1] of integer;  k, t : integer;  k:=0;   t:=0;

cobegin:
PROCESS producer                    PROCESS consumer
begin                               begin
11:     produce item;               12:     p(mutex);
        p(mutex);                           p(full);
        p(empty);                           item = b[t];
        b[k]:=item;                         t:=(t+1) mod n;
        k:=(k+1) mod n;                     v(empty);
        v(full);                            v(mutex);
        v(mutex);                           consume item;
        goto 11;                            goto 12;
end;                                end;
coend;
end;
```

对PV嵌套问题，一般情况下：同步的PV操作在外，互斥的PV操作在内。

可能引起进程死锁

1. 生产者——消费者问题

```
begin
    b: array[0..n-1] of integer;  k, t : integer;  k:=0;   t:=0;
    mutex, full, empty : semaphore; mutex.value:=1; mutex.value:=0; empty.value:=n;
cobegin:
```

PROCESS producer	PROCESS consumer
begin	begin
11: produce item;	12: p(full);
p(empty);	p(mutex);
p(mutex);	item = b[t];
b[k]:=item;	t:=(t+1) mod n;
k:=(k+1) mod n;	v(mutex);
v(mutex);	v(empty);
v(full);	consume item;
goto 11;	goto 12;
end;	end;

```
coend;
end;
```

请大家课后思考：对生产者和消费者的数量有没有约束？

1. 生产者——消费者问题

```
begin
    b: array[0..n-1] of integer;  k, t : integer;  k:=0;   t:=0;
    mutex, full, empty : semaphore; mutex.value:=1; mutex.value:=0; empty.value:=n;
cobegin:
```

PROCESS producer	PROCESS consumer
begin	begin
11: produce item;	12: p(full);
p(empty);	p(mutex);
p(mutex);	item = b[t];
b[k]:=item;	t:=(t+1) mod n;
k:=(k+1) mod n;	v(mutex);
v(mutex);	v(empty);
v(full);	consume item;
goto 11;	goto 12;
end;	end;

```
coend;
end;
```

怎么理解：互斥实质上是同步的一种特殊情况?

1. 生产者——消费者问题

```
begin
    b: array[0..n-1] of integer;  k, t : integer;  k:=0;   t:=0;
    mutex, full, empty : semaphore; mutex.value:=1; mutex.value:=0; empty.value:=n;
cobegin:
PROCESS producer                          PROCESS consumer
begin                                     begin
11:        produce item;                  12:        p(full);
           p(empty);                                 p(mutex);
           p(mutex);                                 item = b[t];
           b[k]:=item;                               t:=(t+1) mod n;
           k:=(k+1) mod n;                           v(mutex);
           v(mutex);                                 v(empty);
           goto 11;                                  consume item;
end;                                      goto 12;
                                          end;
coend;
end;
```

我需要用的互斥资源空闲吗?

互斥资源已经空闲了

怎么理解：互斥实质上是同步的一种特殊情况?

同样可以把mutex看成一个消息

同步：直接的相互制约关系

推进的速度直接相互影响

互斥：间接的相互制约关系

互斥资源的使用情况间接影响

1. 生产者——消费者问题

例： A、B通过信箱辩论，每人从自己信箱中取得对方的问题，并将答案和向对方提出的新问题组成一个邮件放人对方信箱中。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信箱中有x个邮件（0<x<M），B信箱中有y个邮件（0<y<N）。两人的操作过程描述如下：

```
cobegin:
PROCESS A {                          PROCESS B {
  while ( TRUE ) {                     while ( TRUE ) {

      从A的信箱中取出一个邮件;            从B的信箱中取出一个邮件;

      回答问题并提出一个新问题;           回答问题并提出一个新问题;

      将新邮件放入B的信箱;                将新邮件放入A的信箱;

  }                                    }
}                                    }
coend
```

对于A的信箱：
A是消费者，B是生产者

对于B的信箱：
B是消费者，A是生产者

1. 生产者——消费者问题

例： A、B通过信箱辩论，每人从自己信箱中取得对方的问题，并将答案和向对方提出的新问题组成一个邮件放人对方信箱中。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信箱中有x个邮件（0<x<M），B信箱中有y个邮件（0<y<N）。两人的操作过程描述如下：

```
cobegin:
PROCESS A {                              PROCESS B {
  while ( TRUE ) {                         while ( TRUE ) {

      从A的信箱中取出一个邮件;                从B的信箱中取出一个邮件;

      回答问题并提出一个新问题;               回答问题并提出一个新问题;

      将新邮件放入B的信箱;                   将新邮件放入A的信箱;

  }                                        }
}                                        }
coend
```

信箱为共享存储区，需要互斥访问

mutex_A : semaphore;
mutex_B : semaphore;

mutex_A.value : = 1 ;
mutex_A.value : = 1 ;

1. 生产者——消费者问题

例： A、B通过信箱辩论，每人从自己信箱中取得对方的问题，并将答案和向对方提出的新问题组成一个邮件放人对方信箱中。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信箱中有x个邮件（0<x<M），B信箱中有y个邮件（0<y<N）。两人的操作过程描述如下：

cobegin:

```
PROCESS A {                          PROCESS B {
  while ( TRUE ) {                     while ( TRUE ) {

      P(mutex_A);                          P(mutex_B);
      从A的信箱中取出一个邮件;             从B的信箱中取出一个邮件;
      V(mutex_A);                          V(mutex_B);

      回答问题并提出一个新问题;           回答问题并提出一个新问题;

      P(mutex_B);                          P(mutex_A);
      将新邮件放入B的信箱;                 将新邮件放入A的信箱;
      V(mutex_B);                          V(mutex_A);

  }                                    }
}                                    }
```

coend

信箱为共享存储区，需要互斥访问

mutex_A : semaphore;
mutex_B : semaphore;

mutex_A.value : = 1 ;
mutex_A.value : = 1 ;

1. 生产者——消费者问题

例： A、B通过信箱辩论，每人从自己信箱中取得对方的问题，并将答案和向对方提出的新问题组成一个邮件放人对方信箱中。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信箱中有x个邮件（0<x<M），B信箱中有y个邮件（0<y<N）。两人的操作过程描述如下：

cobegin:

```
PROCESS A {                          PROCESS B {
  while ( TRUE ) {                     while ( TRUE ) {

      P(mutex_A);                          P(mutex_B);
      从A的信箱中取出一个邮件;              从B的信箱中取出一个邮件;
      V(mutex_A);                          V(mutex_B);

      回答问题并提出一个新问题;            回答问题并提出一个新问题;

      P(mutex_B);                          P(mutex_A);
      将新邮件放入B的信箱;                  将新邮件放入A的信箱;
      V(mutex_B);                          V(mutex_A);

  }                                    }
}                                    }
```

coend

对A信箱：

消费者A在取走一件产品前，
需要“有一个满单元”的消息
生产者B在放入一个产品后，
发送“有一个满单元”的消息

Full_A : semaphore;

Full_A.value : = x ;

1. 生产者——消费者问题

例： A、B通过信箱辩论，每人从自己信箱中取得对方的问题，并将答案和向对方提出的新问题组成一个邮件放人对方信箱中。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信箱中有x个邮件（0<x<M），B信箱中有y个邮件（0<y<N）。两人的操作过程描述如下：

cobegin:

```
PROCESS A {                          PROCESS B {
  while ( TRUE ) {                     while ( TRUE ) {
      P(Full_A);
      P(mutex_A);                          P(mutex_B);
      从A的信箱中取出一个邮件;             从B的信箱中取出一个邮件;
      V(mutex_A);                          V(mutex_B);

      回答问题并提出一个新问题;           回答问题并提出一个新问题;

      P(mutex_B);                          P(mutex_A);
      将新邮件放入B的信箱;                 将新邮件放入A的信箱;
      V(mutex_B);                          V(mutex_A);
                                           V(Full_A);
  }                                    }
}                                    }
```

coend

对A信箱：

消费者A在取走一件产品前，
需要“有一个满单元”的消息
生产者B在放入一个产品后，
发送“有一个满单元”的消息

Full_A : semaphore;

Full_A.value : = x ;

1. 生产者——消费者问题

例： A、B通过信箱辩论，每人从自己信箱中取得对方的问题，并将答案和向对方提出的新问题组成一个邮件放人对方信箱中。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信箱中有x个邮件（0<x<M），B信箱中有y个邮件（0<y<N）。两人的操作过程描述如下：

cobegin:

```
PROCESS A {                          PROCESS B {
  while ( TRUE ) {                     while ( TRUE ) {
      P(Full_A);
      P(mutex_A);                          P(mutex_B);
      从A的信箱中取出一个邮件;             从B的信箱中取出一个邮件;
      V(mutex_A);                          V(mutex_B);

      回答问题并提出一个新问题;           回答问题并提出一个新问题;

      P(mutex_B);                          P(mutex_A);
      将新邮件放入B的信箱;                 将新邮件放入A的信箱;
      V(mutex_B);                          V(mutex_A);
                                           V(Full_A);
  }                                    }
}                                    }
```

coend

对A信箱：

生产者B在放入一件产品前，
需要“有一个空单元”的消息
消费者A在取走一个产品后，
发送“有一个空单元”的消息

Empty_A : semaphore;

Empty_A.value : = M-x ;

1. 生产者——消费者问题

例： A、B通过信箱辩论，每人从自己信箱中取得对方的问题，并将答案和向对方提出的新问题组成一个邮件放人对方信箱中。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信箱中有x个邮件（0<x<M），B信箱中有y个邮件（0<y<N）。两人的操作过程描述如下：

cobegin:

```
PROCESS A {                          PROCESS B {
  while ( TRUE ) {                     while ( TRUE ) {
      P(Full_A);
      P(mutex_A);                          P(mutex_B);
      从A的信箱中取出一个邮件;             从B的信箱中取出一个邮件;
      V(mutex_A);                          V(mutex_B);
      V(Empty_A);
      回答问题并提出一个新问题;           回答问题并提出一个新问题;
                                           P(Empty_A);
      P(mutex_B);                          P(mutex_A);
      将新邮件放入B的信箱;                 将新邮件放入A的信箱;
      V(mutex_B);                          V(mutex_A);
                                           V(Full_A);
  }                                    }
}                                    }
```

coend

对A信箱：

生产者B在放入一件产品前，
需要“有一个空单元”的消息
消费者A在取走一个产品后，
发送“有一个空单元”的消息

Empty_A : semaphore;

Empty_A.value : = M-x ;

1. 生产者——消费者问题

例： A、B通过信箱辩论，每人从自己信箱中取得对方的问题，并将答案和向对方提出的新问题组成一个邮件放人对方信箱中。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信箱中有x个邮件（0<x<M），B信箱中有y个邮件（0<y<N）。两人的操作过程描述如下：

cobegin:

```
PROCESS A {                          PROCESS B {
  while ( TRUE ) {                     while ( TRUE ) {
      P(Full_A);
      P(mutex_A);                          P(mutex_B);
      从A的信箱中取出一个邮件;             从B的信箱中取出一个邮件;
      V(mutex_A);                          V(mutex_B);
      V(Empty_A);
      回答问题并提出一个新问题;           回答问题并提出一个新问题;
                                           P(Empty_A);
      P(mutex_B);                          P(mutex_A);
      将新邮件放入B的信箱;                 将新邮件放入A的信箱;
      V(mutex_B);                          V(mutex_A);
                                           V(Full_A);
  }                                    }
}                                    }
```

coend

对B信箱：

消费者B在取走一件产品前，
需要“有一个满单元”的消息
生产者A在放入一个产品后，
发送“有一个满单元”的消息

Full_B : semaphore;

Full_B.value : = y ;

1. 生产者——消费者问题

例： A、B通过信箱辩论，每人从自己信箱中取得对方的问题，并将答案和向对方提出的新问题组成一个邮件放人对方信箱中。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信箱中有x个邮件（0<x<M），B信箱中有y个邮件（0<y<N）。两人的操作过程描述如下：

cobegin:

```
PROCESS 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(mutex_B);
      将新邮件放入B的信箱;
      V(mutex_B);
      V(Full_B);
  }
}
```

```
PROCESS B {
  while ( TRUE ) {
      P(Full_B);
      P(mutex_B);
      从B的信箱中取出一个邮件;
      V(mutex_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);
  }
}
```

coend

对B信箱：

消费者B在取走一件产品前，
需要“有一个满单元”的消息
生产者A在放入一个产品后，
发送“有一个满单元”的消息

Full_B : semaphore;

Full_B.value : = y ;

1. 生产者——消费者问题

例： A、B通过信箱辩论，每人从自己信箱中取得对方的问题，并将答案和向对方提出的新问题组成一个邮件放人对方信箱中。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信箱中有x个邮件（0<x<M），B信箱中有y个邮件（0<y<N）。两人的操作过程描述如下：

cobegin:

```
PROCESS A {                          PROCESS B {
  while ( TRUE ) {                     while ( TRUE ) {
      P(Full_A);                           P(Full_B);
      P(mutex_A);                          P(mutex_B);
      从A的信箱中取出一个邮件;             从B的信箱中取出一个邮件;
      V(mutex_A);                          V(mutex_B);
      V(Empty_A);
      回答问题并提出一个新问题;           回答问题并提出一个新问题;
                                           P(Empty_A);
      P(mutex_B);                          P(mutex_A);
      将新邮件放入B的信箱;                 将新邮件放入A的信箱;
      V(mutex_B);                          V(mutex_A);
      V(Full_B);                           V(Full_A);
  }                                    }
}                                    }
```

coend

对B信箱：

生产者A在放入一件产品前，
需要“有一个空单元”的消息
消费者B在取走一个产品后，
发送“有一个空单元”的消息

Empty_B : semaphore;

Empty_B.value : = N-y;

1. 生产者——消费者问题

例： A、B通过信箱辩论，每人从自己信箱中取得对方的问题，并将答案和向对方提出的新问题组成一个邮件放人对方信箱中。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信箱中有x个邮件（0<x<M），B信箱中有y个邮件（0<y<N）。两人的操作过程描述如下：

cobegin:

```
PROCESS 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);
  }
}
```

```
PROCESS 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);
  }
}
```

coend

对B信箱：

生产者A在放入一件产品前，
需要“有一个空单元”的消息
消费者B在取走一个产品后，
发送“有一个空单元”的消息

Empty_B : semaphore;

Empty_B.value : = N-y;

1. 生产者——消费者问题

例： A、B通过信箱辩论，每人从自己信箱中取得对方的问题，并将答案和向对方提出的新问题组成一个邮件放人对方信箱中。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信箱中有x个邮件（0<x<M），B信箱中有y个邮件（0<y<N）。两人的操作过程描述如下：

cobegin:

```
PROCESS 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);
  }
}
```

```
PROCESS 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);
  }
}
```

coend

```
mutex_A, mutex_B : semaphore;
Full_A, Full_B : semaphore;
Empty_A, Empty_B : semaphore;

mutex_A.value:=1;
mutex_B.value:=1;

Full_A,.value:= x;
Full_B.value:= y;

Empty_A := M-x;
Empty_B := N-y;
```

2. 嗜睡理发师问题

1. 理发店有一名理发师，一把理发椅。
2. 若干把客户等待理发的椅子，进入理发店的客户发现没有空余的位置时离开。
3. 在没有顾客光顾时，理发师在椅子上睡觉，等待客户将其唤醒。

2. 嗜睡理发师问题

计数器waiting，记录当前理发店内顾客数。顾客数达到阈值，进程结束而不是挂起。

```
int waiting = 0;
```

```
void barber(void)
{
      while( TRUE ){

          waiting = waiting-1;

          cuthair( );
       }
}
```

```
void customer(void)
{

       if ( waiting < CHAIRS) {
              waiting = waiting+1 ;

              get_haircut( );
        }
       else  {

        }
}
```

```
main()
{
      cobegin
      {
          barber();
          customer( );
       }
}
```

2. 嗜睡理发师问题

计数器waiting，记录当前理发店内顾客数。顾客数达到阈值，进程结束而不是挂起。

```
int waiting = 0;
```

```
void barber(void)
{
      while( TRUE ){

          waiting = waiting-1;

          cuthair( );
       }
}
```

```
void customer(void)
{

       if ( waiting < CHAIRS) {
              waiting = waiting+1 ;

              get_haircut( );
        }
       else  {

        }
}
```

1. 互斥信号量 mutex，保证对waiting的互斥访问

semaphore mutex;
mutex.value : =1;

```
main()
{
      cobegin
      {
          barber();
          customer( );
       }
}
```

2. 嗜睡理发师问题

计数器waiting，记录当前理发店内顾客数。顾客数达到阈值，进程结束而不是挂起。

```
int waiting = 0;
```

```
void barber(void)
{
      while( TRUE ){

          P( mutex );
          waiting = waiting-1;

          V( mutex );
          cuthair( );
       }
}
```

```
void customer(void)
{
      P( mutex );
      if ( waiting < CHAIRS) {
              waiting = waiting+1 ;

              V( mutex );

              get_haircut( );
       }
       else  {
              V( mutex );
       }
}
```

1. 互斥信号量 mutex，保证对waiting的互斥访问

semaphore mutex;
mutex.value : =1;

```
main()
{
      cobegin
      {
          barber();
          customer( );
       }
}
```

2. 嗜睡理发师问题

计数器waiting，记录当前理发店内顾客数。顾客数达到阈值，进程结束而不是挂起。

```
int waiting = 0;
```

```
void barber(void)
{
      while( TRUE ){

          P( mutex );
          waiting = waiting-1;

          V( mutex );
          cuthair( );
      }
}
```

```
void customer(void)
{
      P( mutex );
      if ( waiting < CHAIRS) {
              waiting = waiting+1 ;

              V( mutex );

              get_haircut( );
       }
       else  {
              V( mutex );
       }
}
```

2. 理发师理发前，需要“有顾客等候”的消息；顾客进店后，发出“有顾客等候”的消息

```
semaphore customers;
customers.value : = 0;
```

```
main()
{
      cobegin
      {
          barber();
          customer( );
       }
}
```

2. 嗜睡理发师问题

计数器waiting，记录当前理发店内顾客数。顾客数达到阈值，进程结束而不是挂起。

```
int waiting = 0;
```

```
void barber(void)
{
      while( TRUE ){
          P( customers );
          P( mutex );
          waiting = waiting-1;

          V( mutex );
          cuthair( );
       }
}
```

```
void customer(void)
{
      P( mutex );
      if ( waiting < CHAIRS) {
              waiting = waiting+1 ;
              V( customers );
              V( mutex );

              get_haircut( );
       }
       else  {
              V( mutex );
       }
}
```

2. 理发师理发前，需要“有顾客等候”的消息；顾客进店后，发出“有顾客等候”的消息

```
semaphore customers;
customers.value : = 0;
```

```
main()
{
      cobegin
      {
          barber();
          customer( );
       }
}
```

2. 嗜睡理发师问题

计数器waiting，记录当前理发店内顾客数。顾客数达到阈值，进程结束而不是挂起。

```
int waiting = 0;
```

```
void barber(void)
{
      while( TRUE ){
          P( customers );
          P( mutex );
          waiting = waiting-1;

          V( mutex );
          cuthair( );
       }
}
```

```
void customer(void)
{
      P( mutex );
      if ( waiting < CHAIRS) {
              waiting = waiting+1 ;
              V( customers );
              V( mutex );

              get_haircut( );
       }
       else  {
              V( mutex );
       }
}
```

3. 顾客理发前，需要“有空闲理发师”的消息；顾客理发结束后，发出“有空闲理发师”的消息

```
semaphore barbers;
barbers.value : = 0;
```

```
main()
{
      cobegin
      {
          barber();
          customer( );
       }
}
```

2. 嗜睡理发师问题

计数器waiting，记录当前理发店内顾客数。顾客数达到阈值，进程结束而不是挂起。

```
int waiting = 0;
```

```
void barber(void)
{
      while( TRUE ){
           P( customers );
           P( mutex );
           waiting = waiting-1;
           V( barbers );
           V( mutex );
           cuthair( );
      }
}
```

```
void customer(void)
{
     P( mutex );
     if ( waiting < CHAIRS) {
              waiting = waiting+1 ;
              V( customers );
              V( mutex );
              P( barbers );
              get_haircut( );
      }
      else  {
              V( mutex );
      }
}
```

3. 顾客理发前，需要“有空闲理发师”的消息；顾客理发结束后，发出“有空闲理发师”的消息

```
semaphore barbers;
barbers.value : = 0;
```

```
main()
{
     cobegin
     {
          barber();
          customer( );
      }
}
```

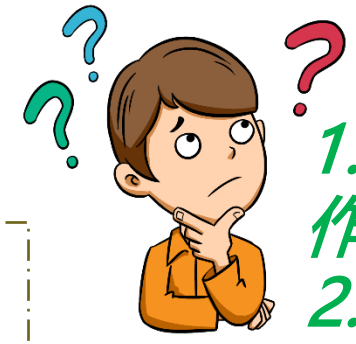
2. 嗜睡理发师问题

计数器waiting，记录当前理发店内顾客数。顾客数达到阈值，进程结束而不是挂起。

```
int waiting = 0;
```

```
void barber(void)
{
      while( TRUE ){
           P( customers );
           P( mutex );
           waiting = waiting-1;
           V( barbers );
           V( mutex );
           cuthair( );
      }
}
```

```
void customer(void)
{
     P( mutex );
     if ( waiting < CHAIRS) {
              waiting = waiting+1 ;
              V( customers );
              V( mutex );
              P( barbers );
              get_haircut( );
      }
      else  {
              V( mutex );
      }
}
```



1. 这里各个p，v操作的位置能否交换？
2. 如果有多个理发师呢？

```
main()
{
      cobegin
      {
          barber();
          customer( );
      }
}
```

3. 读者-写者问题

用于对数据库或数据文件的并发访问建模。**读进程**只进行读操作，不修改数据。**写进程**有可能修改数据。读写进程可能同时存在多个

```
void reader() {


      READUNIT();



}
```

```
void writer()
{
        WRITEUNIT();

}
```

3. 读者-写者问题

用于对数据库或数据文件的并发访问建模。**读进程**只进行读操作，不修改数据。**写进程**有可能修改数据。读写进程可能同时存在多个

```
void reader() {



      READUNIT();



}
```

```
void writer()
{
        P(wmutex);
        WRITEUNIT();
        V(wmutex);
}
```

1. **写进程**需与所有其他写进程互斥访问数据文件。

semaphore wmutex;
wmutex.value : = 1;

3. 读者-写者问题

用于对数据库或数据文件的并发访问建模。**读进程**只进行读操作，不修改数据。**写进程**有可能修改数据。读写进程可能同时存在多个

2. **多个读进程**可同时访问，**写进程**需与所有读进程互斥访问。

```
int readcount = 0;

void reader() {
    readcount++;
    if (readcount == 1) P(wmutex);

    READUNIT();

    readcount--;
    if (readcount == 0) V(wmutex);
}

id writer()
{
    P(wmutex);
    WRITEUNIT();
    V(wmutex);
}
```

第一个进入的读进程通过P操作封锁写进程

最后一个离开的读进程通过V操作释放写进程

3. 读者-写者问题

用于对数据库或数据文件的并发访问建模。**读进程**只进行读操作，不修改数据。**写进程**有可能修改数据。读写进程可能同时存在多个

```
int readcount = 0;
```

```
void reader() {
    p(r);
    readcount++;
    if (readcount == 1) P(wmutex);
    v(r);
    READUNIT();
    p(r);
    readcount--;
    if (readcount == 0) V(wmutex);
    v(r);
}
```

读进程之间互斥的访问计数器readcount

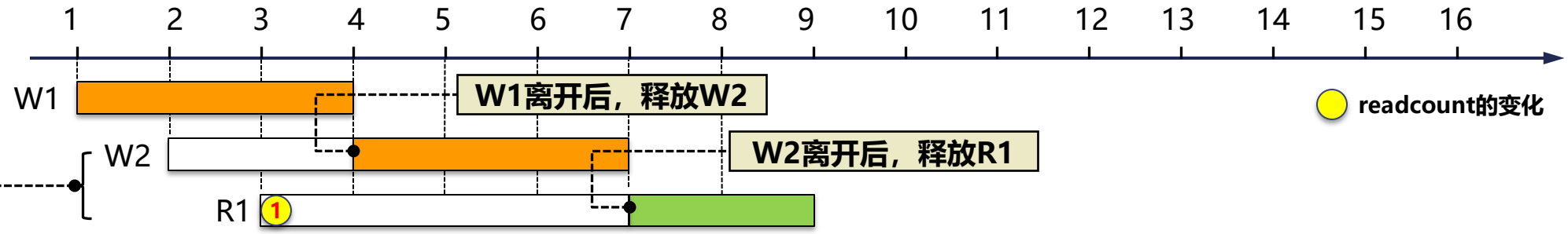
```
void writer()
{
        P(wmutex);
        WRITEUNIT();
        V(wmutex);
}
```

3. readcount是被多个读进程访问的临界资源，需设置互斥信号量。

```
semaphore r;
r.value : = 1;
```

3. 读者-写者问题

假设写进程操作数据库时间为3s，读进程操作数据库时间为2s，如果进程按如下序列以1s的间隔相继到达：W1，W2，R1，R2，R3，W3，R4，R5。



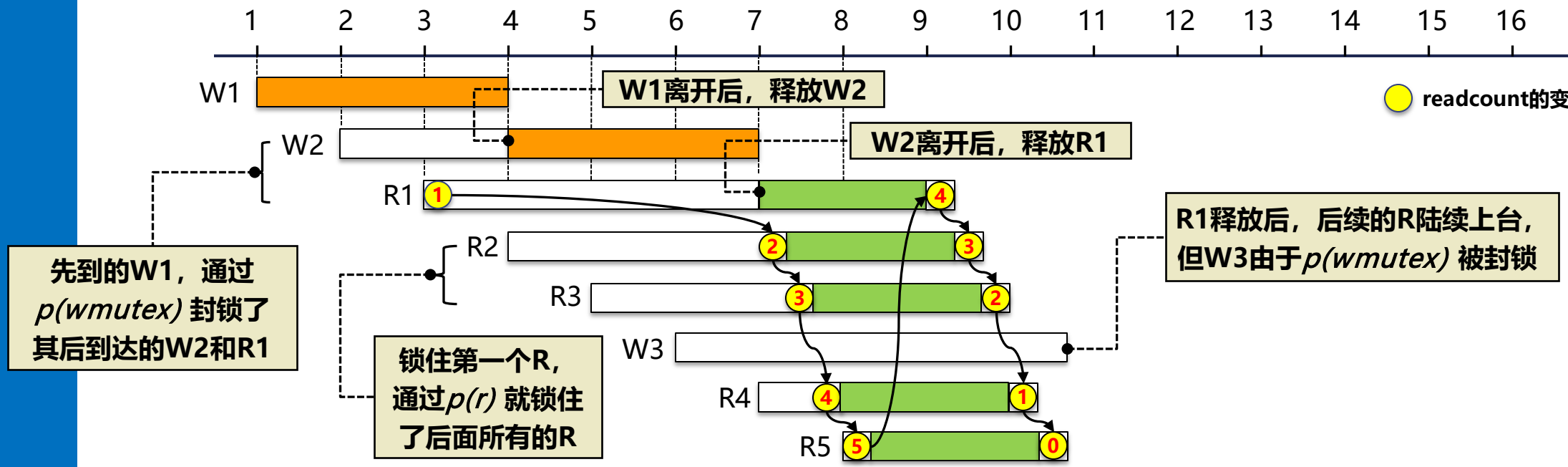
先到的W1，通过 *p(wmutex)* 封锁了其后到达的W2和R1

```
void reader() {
    p(r);
    readcount++;
    if (readcount == 1) P(wmutex);
    v(r);
    READUNIT();
    p(r);
    readcount--;
    if (readcount == 0) V(wmutex);
    v(r);
}
```

```
void writer()
{
    P(wmutex);
    WRITEUNIT();
    V(wmutex);
}
```

3. 读者-写者问题

假设写进程操作数据库时间为3s，读进程操作数据库时间为2s，如果进程按如下序列以1s的间隔相继到达：W1，W2，R1，R2，R3，W3，R4，R5。

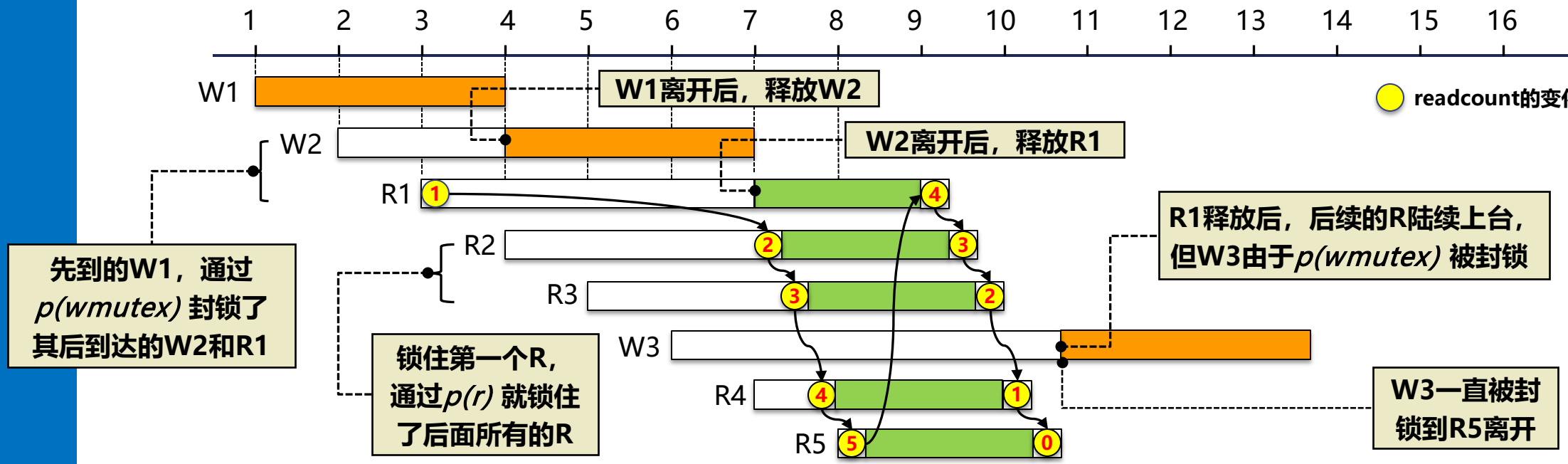



```
void reader() {
    p(r);
    readcount++;
    if (readcount == 1) P(wmutex);
    v(r);
    READUNIT();
    p(r);
    readcount--;
    if (readcount == 0) V(wmutex);
    v(r);
}
```

```
void writer()
{
    P(wmutex);
    WRITEUNIT();
    V(wmutex);
}
```

3. 读者-写者问题

假设写进程操作数据库时间为3s，读进程操作数据库时间为2s，如果进程按如下序列以1s的间隔相继到达：W1，W2，R1，R2，R3，W3，R4，R5。

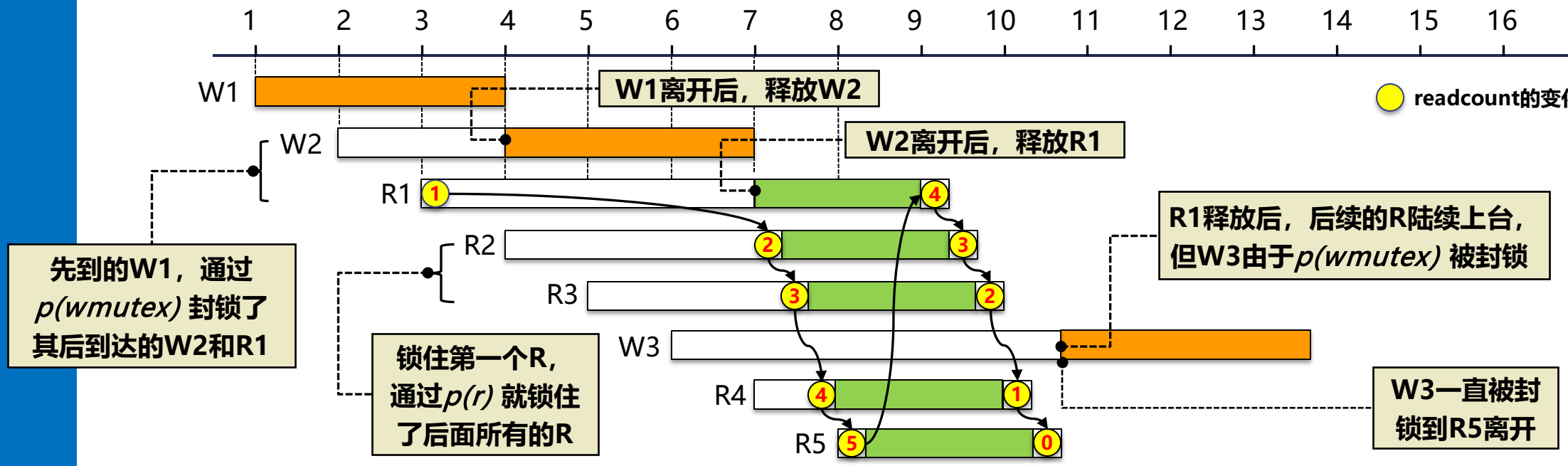



```
void reader() {
    p(r);
    readcount++;
    if (readcount == 1) P(wmutex);
    v(r);
    READUNIT();
    p(r);
    readcount--;
    if (readcount == 0) V(wmutex);
    v(r);
}
```

```
void writer()
{
    P(wmutex);
    WRITEUNIT();
    V(wmutex);
}
```

3. 读者-写者问题

假设写进程操作数据库时间为3s，读进程操作数据库时间为2s，如果进程按如下序列以1s的间隔相继到达：W1，W2，R1，R2，R3，W3，R4，R5。




```
void reader() {
    p(r);
    readcount++;
    if (readcount == 1) P(wmutex);
    v(r);
    READUNIT();
    p(r);
    readcount--;
    if (readcount == 0) V(wmutex);
    v(r);
}
```

```
void writer()
{
    P(wmutex);
    WRITEUNIT();
    V(wmutex);
}
```

在某个读者进程访问数据区时，只要有读请求，写操作将延迟到系统中所有读请求（包括写操作之后出现的读请求）全部得到满足之后。因此写操作可能会“饿死”。

3. 读者-写者问题

```
void reader() {
    while(true) {
        p(rmutex);
        p(r);
        readcount++;
        if (readcount == 1) p(wmutex);
        v(r);
        v(rmutex);

        READUNIT();

        p(r);
        readcount--;
        if (readcount == 0) v(wmutex);
        v(r);
    }
}
```

```
void writer() {
    while(true)   {
        p(w);
        writecount++;
        if(writecount == 1) p(rmutex);
        v(w);

        p(wmutex);
        WRITEUNIT();
        v(wmutex);

        p(w);
        writecount--;
        if(writecount == 0) v(rmutex);
        v(w);
    }
}
```

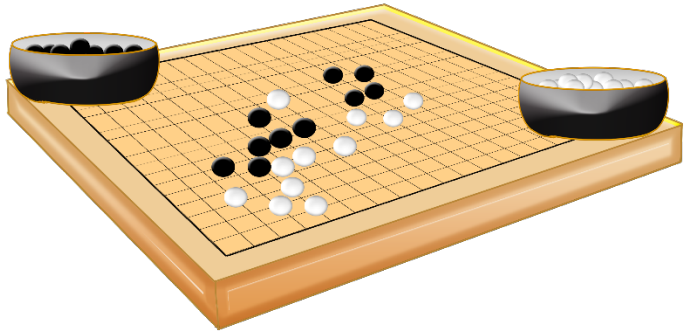
写进程优先级高

是否存在读者写者相对公平的解决方案?

4. 黑白棋问题

问题描述：两个人下棋，一方执黑棋，一方执白棋。要求双方轮流下子。
给出两种情况的解决办法：（1）执黑子一方先下

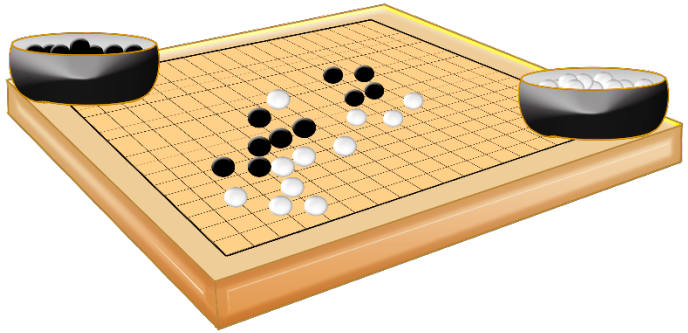
```
begin

cobegin:
PROCESS Black {                    PROCESS White {
   while ( 没结束 ) {                  while ( 没结束 ) {

      下一黑子;                          下一白子;

   }                                  }
}                                  }
coend
end
```



4. 黑白棋问题

问题描述：两个人下棋，一方执黑棋，一方执白棋。要求双方轮流下子。
给出两种情况的解决办法：（1）执黑子一方先下

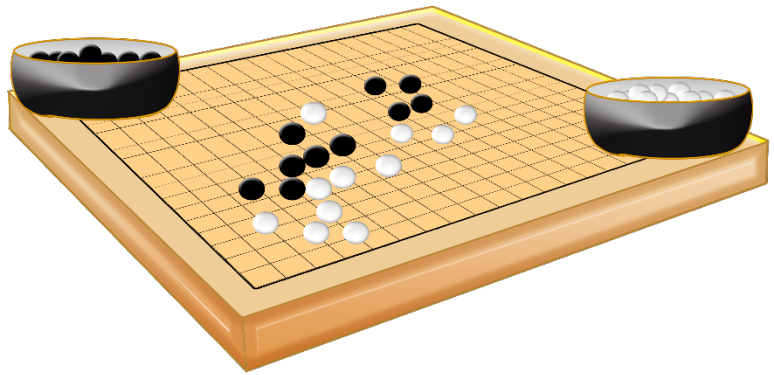
```
begin
black, white : semaphore; black.value:=1; white.value:=0
cobegin:
 PROCESS Black {                    PROCESS White {
   while ( 没结束 ) {                   while ( 没结束 ) {
     p(black);                          p(white);
     下一黑子;                          下一白子;
     v(white);                          v(black);
   }                                  }
}                                   }
coend
end
```



4. 黑白棋问题

问题描述：两个人下棋，一方执黑棋，一方执白棋。要求双方轮流下子。给出两种情况的解决办法：（2）双方都可先下，谁先抢到棋盘谁先下。然后轮流。

```
begin
m: semaphore; m.value:=1; int turn = 0;
cobegin:
 PROCESS Black {                    PROCESS White {
   while ( 没结束 ) {                   while ( 没结束 ) {
     p(m);                              p(m);
     if (turn <> 2)   下一黑子;         if (turn <> 1)   下一白子;
     turn = 2;                          turn = 1;
     v(m);                              v(m);
    }                                  }
}                                   }
coend
end
```



本节小结

1　利用信号量机制解决经典的进程通信问题

阅读讲义57页~62页

认真考虑课件中的问题，尝试解决方案

E06：并发进程（进程通信）