

计算机操作系统

Operating Systems

田卫东

March, 2014

第2章 进程管理

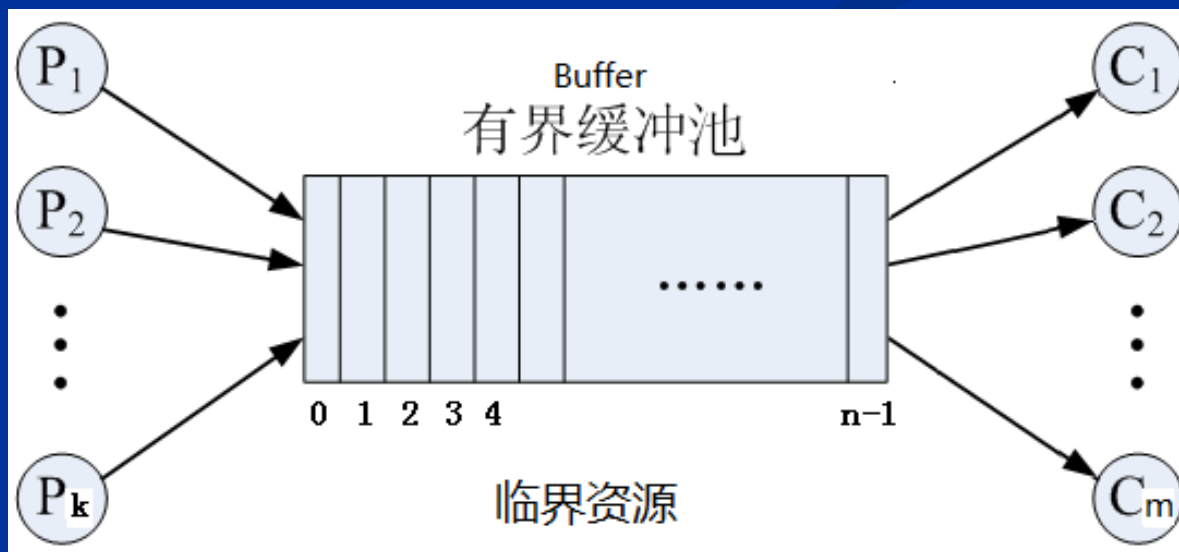
2.4 经典进程同步问题

2.4.1 生产者-消费者问题

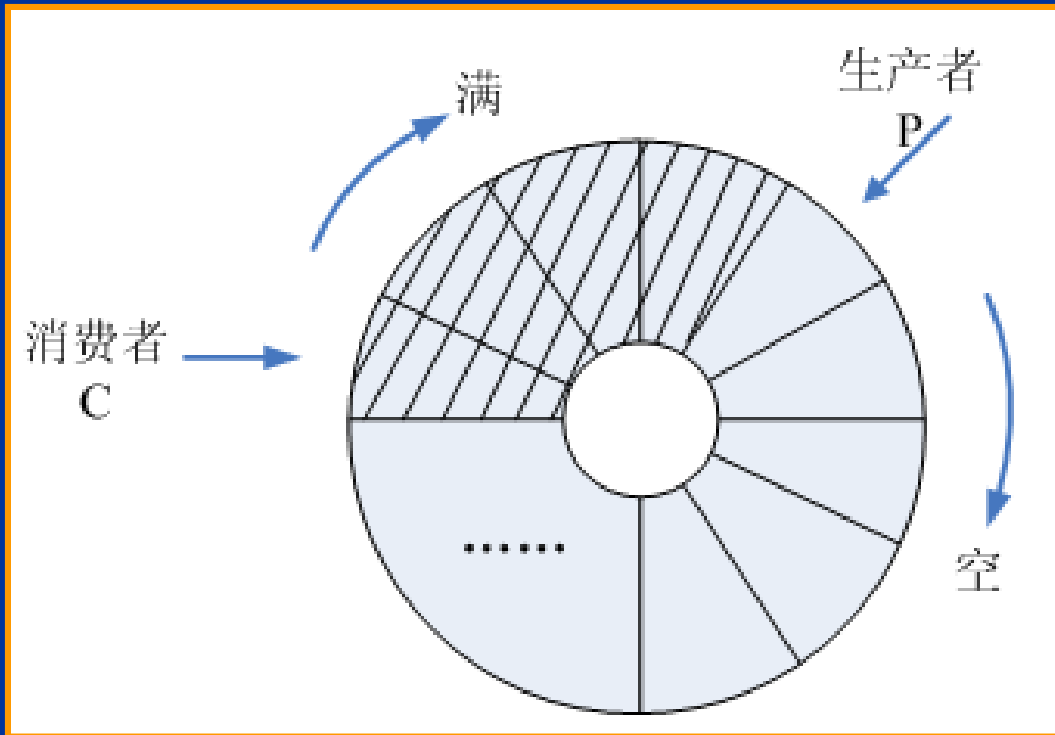
(1) 生产者消费者问题

多个生产者、多个消费者通过共享含 n 个缓冲区的缓冲池Buffer协作。其中生产者负责生产数据并投入缓冲池，消费者从缓冲池中取数据消费，生产者和消费者，每次生产/消费1个数据，要求每个数据必须且只被消费一次。缓冲池为临界资源。

请用记录型信号量进行同步。

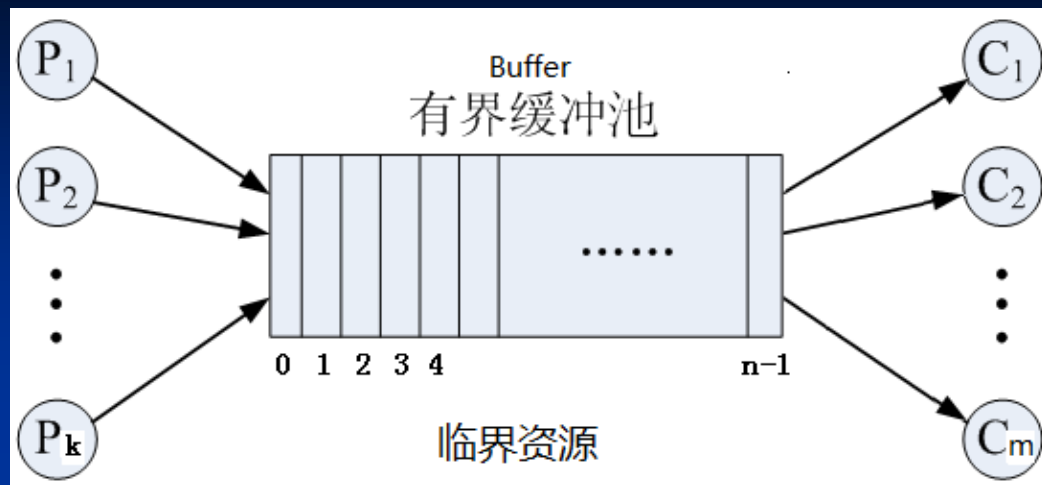


■ 有界缓冲池 --> 循环缓冲区



- 循环队列
- 队列数据结构
- 入队操作
- 出队操作

■ 进程资源共享关系和同步关系分析



■ 生产者进程：空缓冲区资源

■ 消费者进程：满缓冲区资源

```

VAR
    in,out: integer := 0, 0 ;
    Buffer: array [0..n-1] of item ;

```

Parbegin

Producer:

begin

repeat

produce an item in nextp ;

Buffer(in) := nextp ;

in := (in + 1) mod n ;

until false

end

Consumer:

begin

repeat

nextc = Buffer(out);

out := (out + 1) mod n ;

consume the item nextc ;

until false

end

Parend

数据结构

入队

出队

```

VAR empty, full: semaphore := n, 0 ;

```

```

    in,out: integer := 0, 0 ;

```

```

    Buffer: array [0..n-1] of item ;

```

Parbegin

Producer:

begin

repeat

produce an item in nextp ;

wait(empty) ;

Buffer(in) := nextp ;

in := (in + 1) mod n ;

signal(full) ;

until false

end

Consumer:

begin

repeat

wait(full) ;

nextc = Buffer(out);

out := (out + 1) mod n ;

signal(empty) ;

consume the item nextc ;

until false

end

Parend

```

VAR mutex, empty, full: semaphore := 1, n, 0 ;
    in,out: integer := 0, 0 ;
    Buffer: array [0..n-1] of item ;
Parbegin
    Producer:
    begin
        repeat
            produce an item in nextp ;
            wait( empty ) ;
            wait( mutex ) ;
            Buffer(in) := nextp ;
            in := (in + 1) mod n ;
            signal( mutex ) ;
            signal( full ) ;
        until false
    end

    Consumer:
    begin
        repeat
            wait( full ) ;
            wait( mutex ) ;
            nextc = Buffer(out);
            out := (out + 1) mod n ;
            signal( mutex ) ;
            signal( empty ) ;
            consume the item nextc ;
        until false
    end
end
Parend

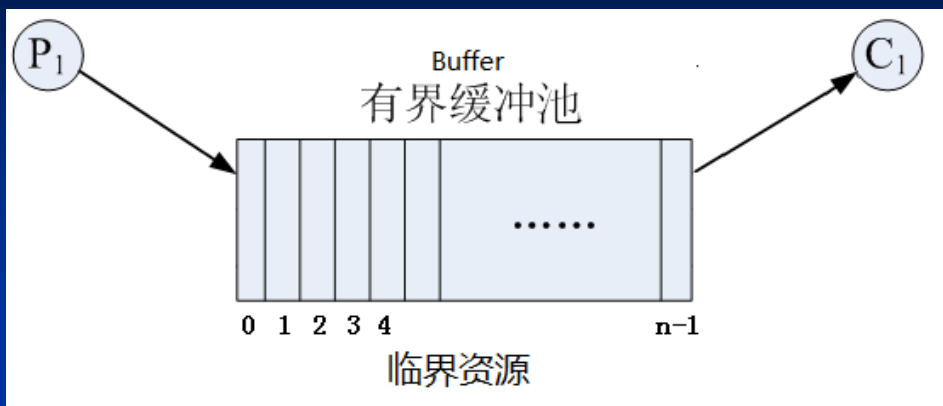
```

(2) 问题思考

- 单生产者单消费者、多缓冲时
- 多生产者多消费者、单缓冲时
- 单生产者单消费者、单缓冲时
- 允许生产者写时，消费者可读
- 当缓冲区无限大；
- 每个消息都要每个消费者消费1次
- 调整生产者wait顺序；
- 调整消费者wait顺序；
- 调整signal顺序；

(2) 问题思考

■ 单生产者单消费者、多缓冲



■ 生产者-消费者标准程序

```
VAR mutex, empty, full: semaphore := 1, n, 0 ;  
    in, out: integer := 0, 0 ;  
    Buffer: array [0..n-1] of item ;
```

```
Parbegin
```

```
    Producer:
```

```
    begin
```

```
        repeat
```

```
            produce an item in nextp ;
```

```
            wait( empty ) ;
```

```
            wait( mutex ) ;
```

```
            Buffer(in) := nextp ;
```

```
            in := (in + 1) mod n ;
```

```
            signal( mutex ) ;
```

```
            signal( full ) ;
```

```
        until false
```

```
    end
```

```
    Consumer:
```

```
    begin
```

```
        repeat
```

```
            wait( full ) ;
```

```
            wait( mutex ) ;
```

```
            nextc = Buffer(out);
```

```
            out := (out + 1) mod n ;
```

```
            signal( mutex ) ;
```

```
            signal( empty ) ;
```

```
            consume the item nextc ;
```

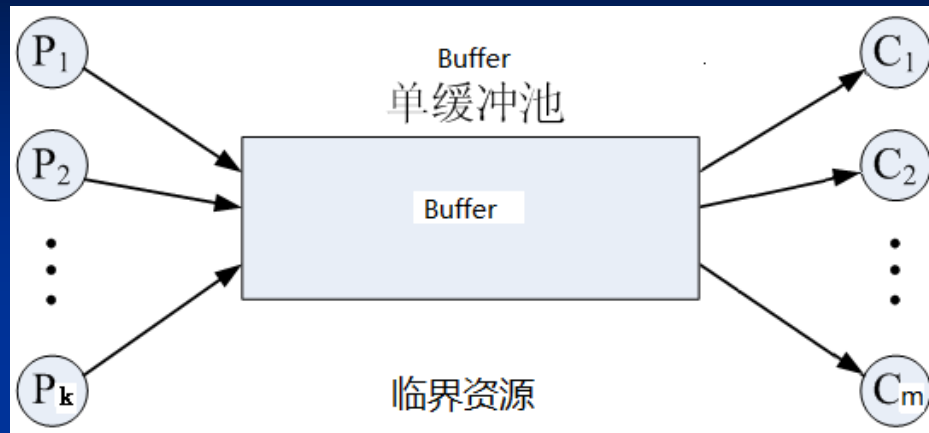
```
        until false
```

```
    end
```

```
Parend
```


(2) 问题思考

■ 多生产者多消费者、单缓冲



■ 生产者-消费者标准程序

```
VAR mutex, empty, full: semaphore := 1, n, 0 ;  
    in, out: integer := 0, 0 ;  
    Buffer: array [0..n-1] of item ;
```

Parbegin

 Producer:

 begin

 repeat

 produce an item in nextp ;

 wait(empty) ;

 wait(mutex) ;

 Buffer(in) := nextp ;

 in := (in + 1) mod n ;

 signal(mutex) ;

 signal(full) ;

 until false

 end

 Consumer:

 begin

 repeat

 wait(full) ;

 wait(mutex) ;

 nextc = Buffer(out);

 out := (out + 1) mod n ;

 signal(mutex) ;

 signal(empty) ;

 consume the item nextc ;

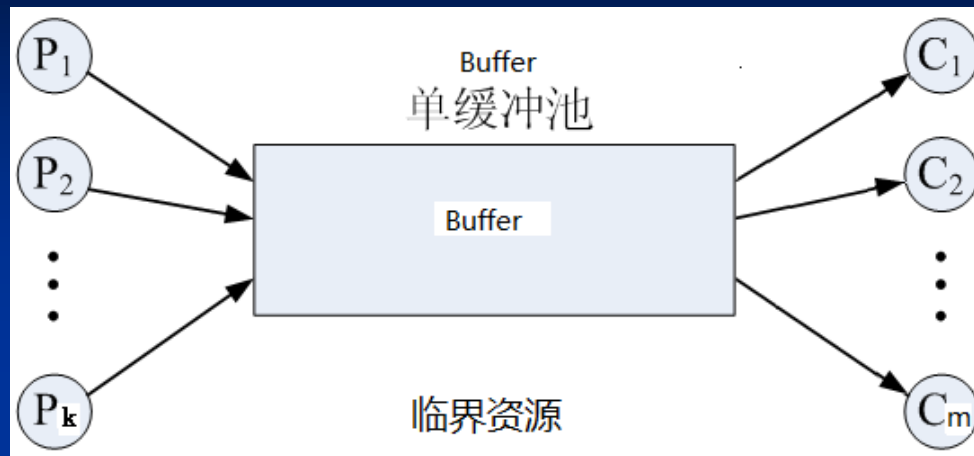
 until false

 end

Parend

(2) 问题思考

■ 多生产者多消费者、单缓冲



```
VAR empty, full: semaphore := 1, 0 ;  
    in, out: integer := 0, 0 ;  
    Buffer: array [0..n-1] of item ;
```

```
Parbegin
```

```
  Producer:
```

```
  begin
```

```
    repeat
```

```
      produce an item in nextp ;
```

```
      wait( empty ) ;
```

```
      Buffer(in) := nextp ;
```

```
      in := (in + 1) mod n ;
```

```
      signal( full ) ;
```

```
    until false
```

```
  end
```

```
  Consumer:
```

```
  begin
```

```
    repeat
```

```
      wait( full ) ;
```

```
      nextc = Buffer(out);
```

```
      out := (out + 1) mod n ;
```

```
      signal( empty ) ;
```

```
      consume the item nextc ;
```

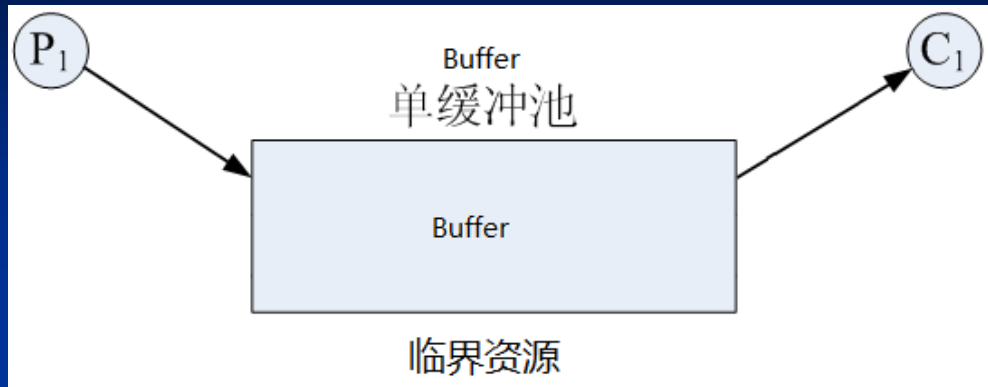
```
    until false
```

```
  end
```

```
Parend
```

(2) 问题思考

■ 单生产者单消费者、单缓冲

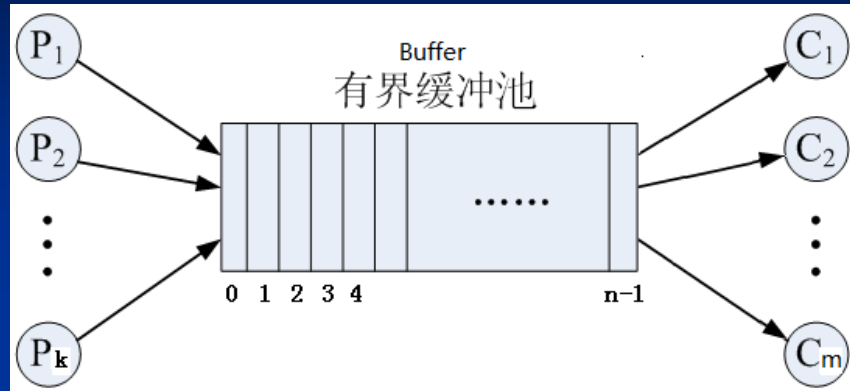


■ 同例3

```
VAR empty, full: semaphore := 1, 0 ;  
    in, out: integer := 0, 0 ;  
    Buffer: array [0..n-1] of item ;  
Parbegin  
    Producer:  
    begin  
        repeat  
            produce an item in nextp ;  
            wait( empty ) ;  
            Buffer(in) := nextp ;  
            in := (in + 1) mod n ;  
            signal( full ) ;  
        until false  
    end  
  
    Consumer:  
    begin  
        repeat  
            wait( full ) ;  
            nextc = Buffer(out);  
            out := (out + 1) mod n ;  
            signal( empty ) ;  
            consume the item nextc ;  
        until false  
    end  
Parend
```

(2) 问题思考

■ 允许生产者写时，消费者可读



■ 生产者-消费者标准程序

```
VAR mutex, empty, full: semaphore := 1, n, 0 ;  
    in, out: integer := 0, 0 ;  
    Buffer: array [0..n-1] of item ;
```

```
Parbegin
```

```
    Producer:
```

```
    begin
```

```
        repeat
```

```
            produce an item in nextp ;
```

```
            wait( empty ) ;
```

```
            wait( mutex ) ;
```

```
            Buffer(in) := nextp ;
```

```
            in := (in + 1) mod n ;
```

```
            signal( mutex ) ;
```

```
            signal( full ) ;
```

```
        until false
```

```
    end
```

```
    Consumer:
```

```
    begin
```

```
        repeat
```

```
            wait( full ) ;
```

```
            wait( mutex ) ;
```

```
            nextc = Buffer(out);
```

```
            out := (out + 1) mod n ;
```

```
            signal( mutex ) ;
```

```
            signal( empty ) ;
```

```
            consume the item nextc ;
```

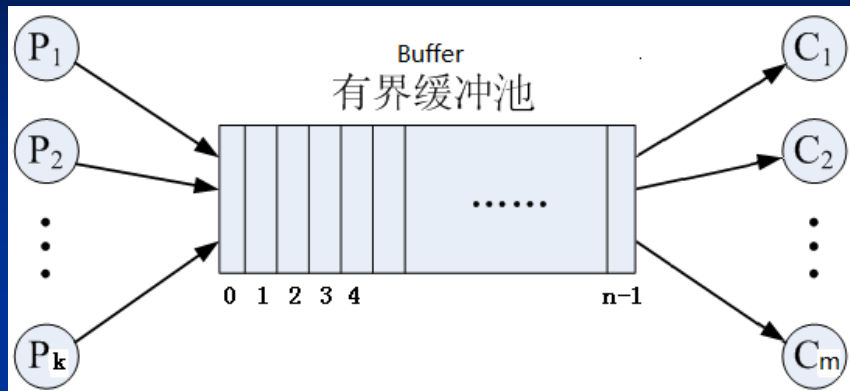
```
        until false
```

```
    end
```

```
Parend
```

(2) 问题思考

■ 允许生产者写时，消费者可读



```
VAR mutexC, mutexP, empty, full: semaphore := 1, 1, n, 0 ;  
    in, out: integer := 0, 0 ;  
    Buffer: array [0..n-1] of item ;
```

```
Parbegin
```

```
    Producer:
```

```
    begin
```

```
        repeat
```

```
            produce an item in nextp ;
```

```
            wait( empty ) ;
```

```
            wait( mutexP ) ;
```

```
            Buffer(in) := nextp ;
```

```
            in := (in + 1) mod n ;
```

```
            signal( mutexP ) ;
```

```
            signal( full ) ;
```

```
        until false
```

```
    end
```

```
    Consumer:
```

```
    begin
```

```
        repeat
```

```
            wait( full ) ;
```

```
            wait( mutexC ) ;
```

```
            nextc = Buffer(out);
```

```
            out := (out + 1) mod n ;
```

```
            signal( mutexC ) ;
```

```
            signal( empty ) ;
```

```
            consume the item nextc ;
```

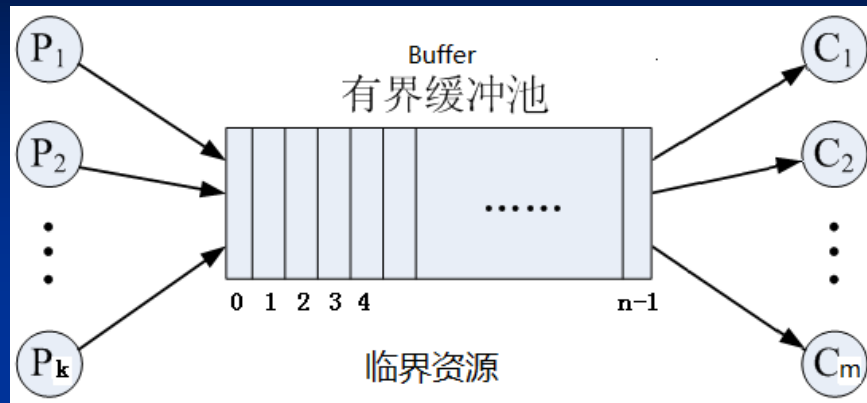
```
        until false
```

```
    end
```

```
Parend
```

(2) 问题思考

■ 缓冲池无限大



■ 生产者-消费者标准程序

```
VAR mutex, empty, full: semaphore := 1, n, 0 ;  
    in, out: integer := 0, 0 ;  
    Buffer: array [0..n-1] of item ;
```

Parbegin

 Producer:

 begin

 repeat

 produce an item in nextp ;

 wait(empty) ;

 wait(mutex) ;

 Buffer(in) := nextp ;

 in := (in + 1) mod n ;

 signal(mutex) ;

 signal(full) ;

 until false

 end

 Consumer:

 begin

 repeat

 wait(full) ;

 wait(mutex) ;

 nextc = Buffer(out);

 out := (out + 1) mod n ;

 signal(mutex) ;

 signal(empty) ;

 consume the item nextc ;

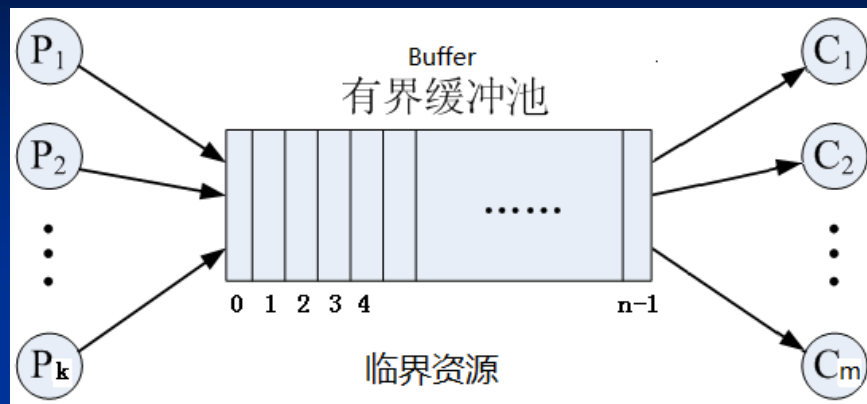
 until false

 end

Parend

(2) 问题思考

■ 缓冲池无限大



```
VAR mutex, full: semaphore := 1, 0 ;
```

```
in,out: integer := 0, 0 ;
```

```
Buffer: array [0..n-1] of item ;
```

```
Parbegin
```

```
  Producer:
```

```
  begin
```

```
    repeat
```

```
      produce an item in nextp ;
```

```
      wait( mutex ) ;
```

```
      Buffer(in) := nextp ;
```

```
      in := (in + 1) mod n ;
```

```
      signal( mutex ) ;
```

```
      signal( full ) ;
```

```
    until false
```

```
  end
```

```
  Consumer:
```

```
  begin
```

```
    repeat
```

```
      wait( full ) ;
```

```
      wait( mutex ) ;
```

```
      nextc = Buffer(out);
```

```
      out := (out + 1) mod n ;
```

```
      signal( mutex ) ;
```

```
      consume the item nextc ;
```

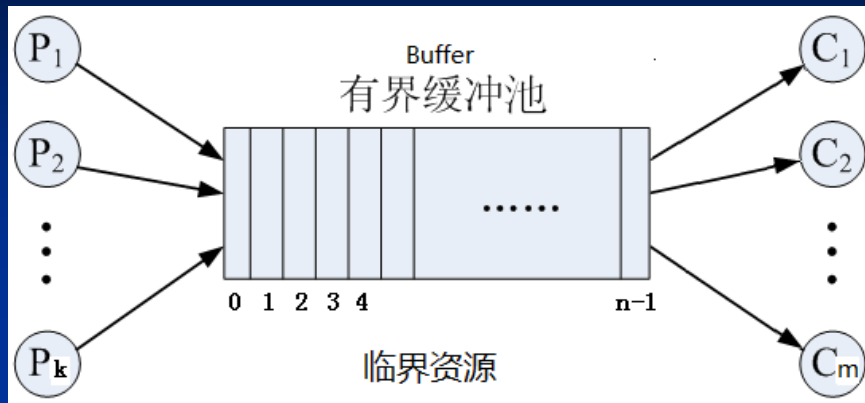
```
    until false
```

```
  end
```

```
Parend
```

(2) 问题思考

■ 调整生产者wait顺序



```
VAR mutex, empty, full: semaphore := 1, n, 0 ;
```

```
in,out: integer := 0, 0 ;
```

```
Buffer: array [0..n-1] of item ;
```

```
Parbegin
```

```
  Producer:
```

```
  begin
```

```
    repeat
```

```
      produce an item in nextp ;
```

```
      wait( mutex ) ;
```

```
      wait( empty ) ;
```

```
      Buffer(in) := nextp ;
```

```
      in := (in + 1) mod n ;
```

```
      signal( mutex ) ;
```

```
      signal( full ) ;
```

```
    until false
```

```
  end
```

```
  Consumer:
```

```
  begin
```

```
    repeat
```

```
      wait( full ) ;
```

```
      wait( mutex ) ;
```

```
      nextc = Buffer(out);
```

```
      out := (out + 1) mod n ;
```

```
      signal( mutex ) ;
```

```
      signal( empty ) ;
```

```
      consume the item nextc ;
```

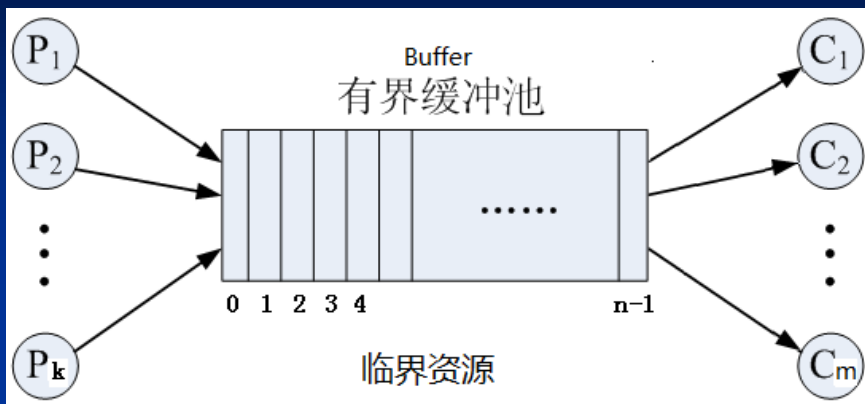
```
    until false
```

```
  end
```

```
Parend
```


(2) 问题思考

■ 调整消费者wait顺序

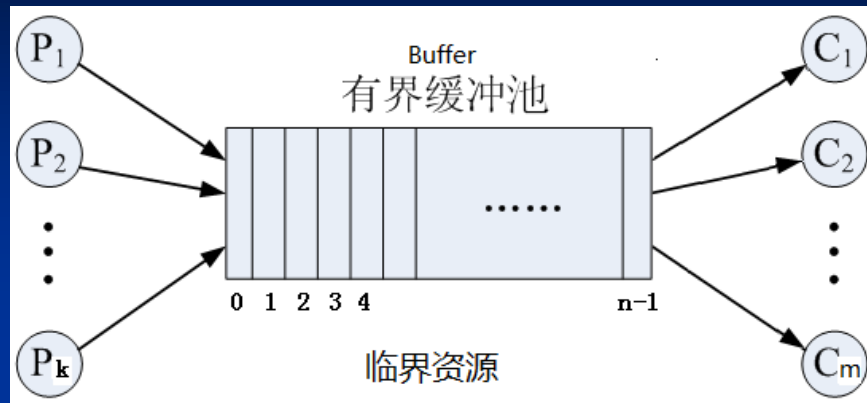


```
VAR mutex, empty, full: semaphore := 1, n, 0 ;  
    in, out: integer := 0, 0 ;  
    Buffer: array [0..n-1] of item ;
```

```
Parbegin  
  Producer:  
  begin  
    repeat  
      produce an item in nextp ;  
      wait( empty ) ;  
      wait( mutex ) ;  
      Buffer(in) := nextp ;  
      in := (in + 1) mod n ;  
      signal( mutex ) ;  
      signal( full ) ;  
    until false  
  end  
  Consumer:  
  begin  
    repeat  
      wait( mutex ) ;  
      wait( full ) ;  
      nextc = Buffer(out);  
      out := (out + 1) mod n ;  
      signal( mutex ) ;  
      signal( empty ) ;  
      consume the item nextc ;  
    until false  
  end  
Parend
```

(2) 问题思考

■ 调整wait顺序

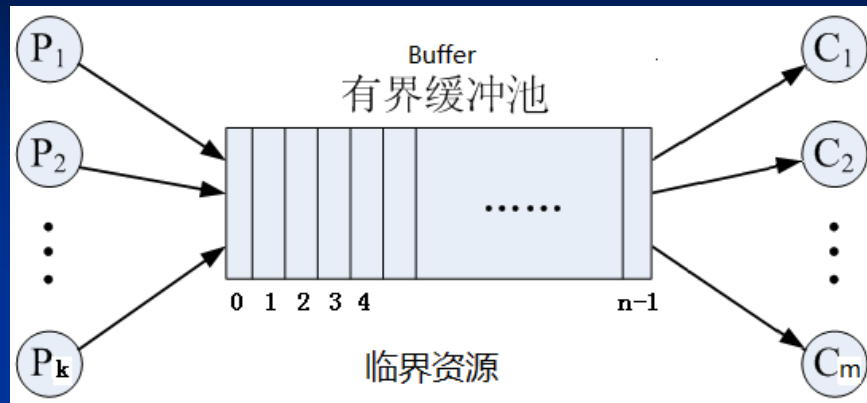


```
VAR mutex, empty, full: semaphore := 1, n, 0 ;  
    in, out: integer := 0, 0 ;  
    Buffer: array [0..n-1] of item ;
```

```
Parbegin  
  Producer:  
  begin  
    repeat  
      produce an item in nextp ;  
      wait( mutex ) ;  
      wait( empty ) ;  
      Buffer(in) := nextp ;  
      in := (in + 1) mod n ;  
      signal( mutex ) ;  
      signal( full ) ;  
    until false  
  end  
  Consumer:  
  begin  
    repeat  
      wait( mutex ) ;  
      wait( full ) ;  
      nextc = Buffer(out);  
      out := (out + 1) mod n ;  
      signal( mutex ) ;  
      signal( empty ) ;  
      consume the item nextc ;  
    until flase  
  end  
Parend
```

(2) 问题思考

■ 调整signal顺序



```
VAR mutex, empty, full: semaphore := 1, n, 0 ;  
    in, out: integer := 0, 0 ;  
    Buffer: array [0..n-1] of item ;
```

```
Parbegin
```

```
    Producer:
```

```
    begin
```

```
        repeat
```

```
            produce an item in nextp ;
```

```
            wait( empty ) ;
```

```
            wait( mutex ) ;
```

```
            Buffer(in) := nextp ;
```

```
            in := (in + 1) mod n ;
```

```
            signal( mutex ) ;
```

```
            signal( full ) ;
```

```
        until false
```

```
    end
```

```
    Consumer:
```

```
    begin
```

```
        repeat
```

```
            wait( full ) ;
```

```
            wait( mutex ) ;
```

```
            nextc = Buffer(out);
```

```
            out := (out + 1) mod n ;
```

```
            signal( empty ) ;
```

```
            signal( mutex ) ;
```

```
            consume the item nextc ;
```

```
        until false
```

```
    end
```

```
Parend
```

生产者-消费者标准程序

```
VAR mutex, empty, full: semaphore := 1, n, 0 ;  
    in, out: integer := 0, 0 ;  
    Buffer: array [0..n-1] of item ;
```

Parbegin

 Producer:

 begin

 repeat

 produce an item in nextp ;

 wait(empty) ;

 wait(mutex) ;

 Buffer(in) := nextp ;

 in := (in + 1) mod n ;

 signal(mutex) ;

 signal(full) ;

 until false

 end

 Consumer:

 begin

 repeat

 wait(full) ;

 wait(mutex) ;

 nextc = Buffer(out);

 out := (out + 1) mod n ;

 signal(mutex) ;

 signal(empty) ;

 consume the item nextc ;

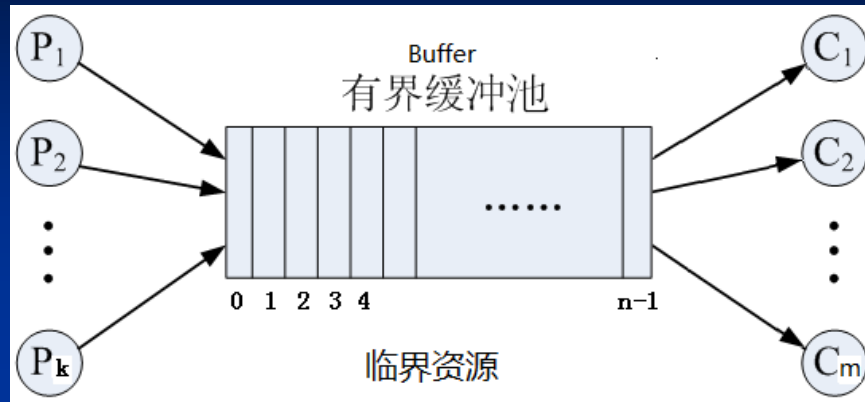
 until false

 end

Parend

(2) 问题思考

每个消费者消费每个消息1次



如何实现？

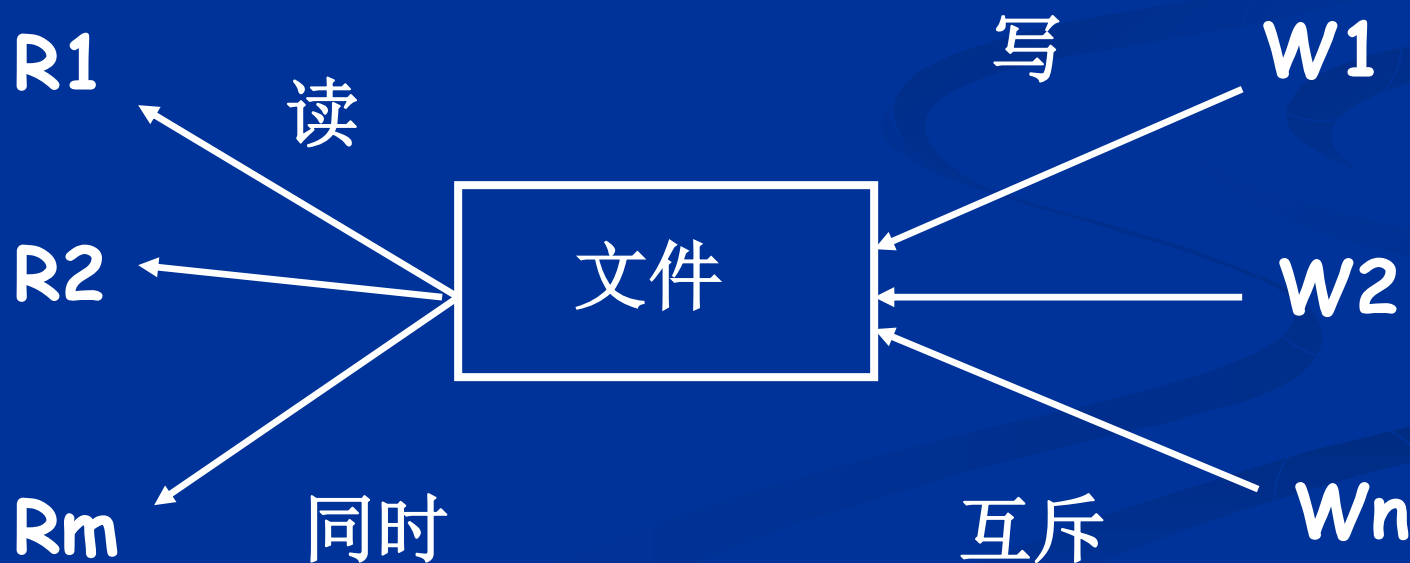
留给大家课后解决

2.4 经典进程同步问题

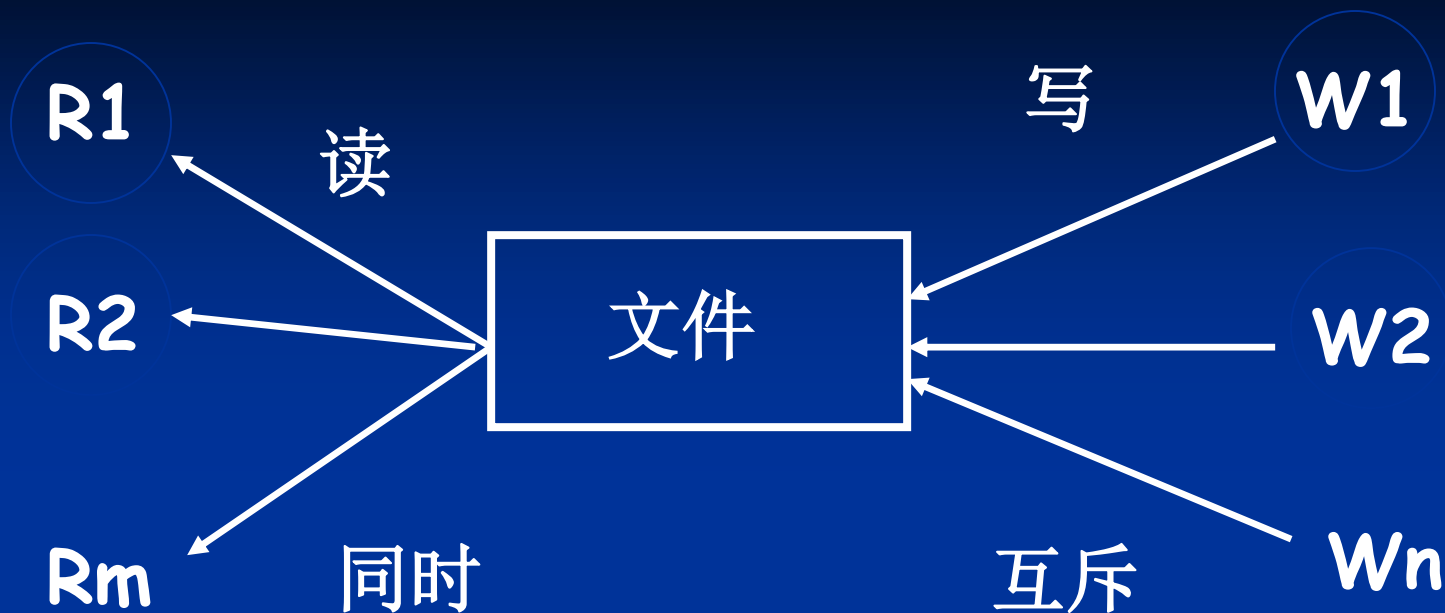
2.4.2 读者-写者问题

(1) 读者写者问题

多个读者、多个写者共享资源（例如文件、数据等），允许多个读者同时访问资源，写者写时不允许其他进程读或写（互斥访问资源）。请用记录型信号量进行同步。



■ 进程资源共享关系和同步关系分析



- 读者进程：不控制
- 写者进程：互斥
- 读者写者顺序：无

■ 读者-写者并发程序实现

```
VAR wmutex: semaphore := 1 ;
Parbegin
  Reader:
  begin
    repeat
      wait( wmutex ) ;
      perform read operation ;
      signal( wmutex ) ;
    until false
  end
  Writer:
  begin
    repeat
      wait( wmutex ) ;
      perform write operation;
      signal( wmutex ) ;
    until false
  end
Parend
```

■ 什么问题？

■ 读者-写者并发程序实现

```
VAR wmutex: semaphore := 1 ;  
    readcount: integer := 0 ;  
Parbegin  
  Reader:  
  begin  
    repeat  
      if ( readcount = 0 ) wait( wmutex ) ;  
      readcount := readcount + 1 ;  
      perform read operation ;  
      readcount := readcount - 1 ;  
      if ( readcount = 0 ) signal( wmutex ) ;  
    until false  
  end  
  Writer:  
  begin  
    repeat  
      wait( wmutex ) ;  
      perform write operation ;  
      signal( wmutex ) ;  
    until false  
  end  
Parend
```

■ 什么问题？

■ 读者-写者并发程序实现

```
VAR wmutex: semaphore := 1 ;
    readcount: integer := 0 ;
Parbegin
  Reader:
  begin
    repeat
      if ( readcount = 0 ) wait( wmutex ) ;
      readcount := readcount + 1 ;
      perform read operation ;
      readcount := readcount - 1 ;
      if ( readcount = 0 ) signal( wmutex ) ;
    until false
  end
  Writer:
  begin
    repeat
      wait( wmutex ) ;
      perform write operation;
      signal( wmutex ) ;
    until flase
  end
Parend
```

■ 读者-写者标准程序

```
VAR rmutex,wmutex: semaphore := 1, 1 ;
    readcount: integer := 0 ;
Parbegin
  Reader:
  begin
    repeat
      wait( rmutex ) ;
      if ( readcount = 0 ) wait( wmutex ) ;
      readcount := readcount + 1 ;
      signal( rmutex ) ;
      perform read operation ;
      wait( rmutex ) ;
      readcount := readcount - 1 ;
      if ( readcount = 0 ) signal( wmutex ) ;
      signal( rmutex ) ;
    until false
  end
  Writer:
  begin
    repeat
      wait( wmutex ) ;
      perform write operation;
      signal( wmutex ) ;
    until flase
  end
Parend
```

■ 读者-写者标准程序

```
VAR rmutex, wmutex: semaphore := 1, 1 ;
    readcount: integer := 0 ;
Parbegin
  Reader:
  begin
    repeat
      wait( rmutex ) ;
      if ( readcount = 0 ) wait( wmutex ) ;
      readcount := readcount + 1 ;
      signal( rmutex ) ;
      perform read operation ;
      wait( rmutex ) ;
      readcount := readcount - 1 ;
      if ( readcount = 0 ) signal( wmutex ) ;
      signal( rmutex ) ;
    until false
  end
  Writer:
  begin
    repeat
      wait( wmutex ) ;
      perform write operation;
      signal( wmutex ) ;
    until false
  end
Parend
```

(2) 标准程序的2个问题

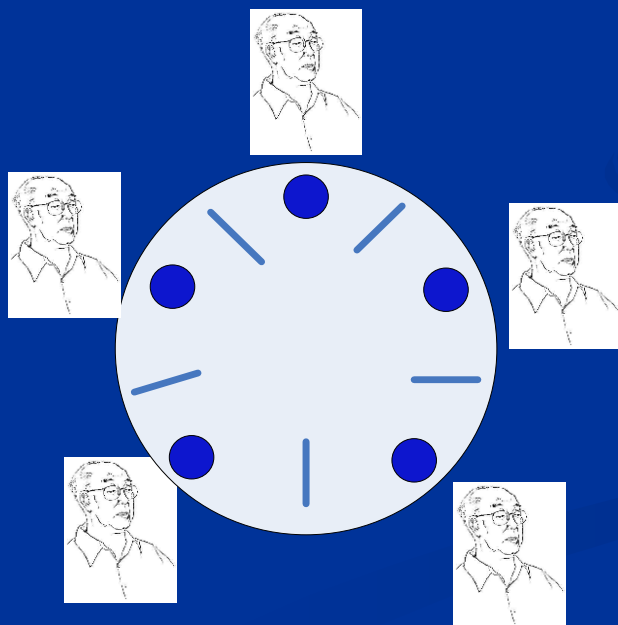
- 执行wait和signal的进程
- 临界区设置

2.4 经典进程同步问题

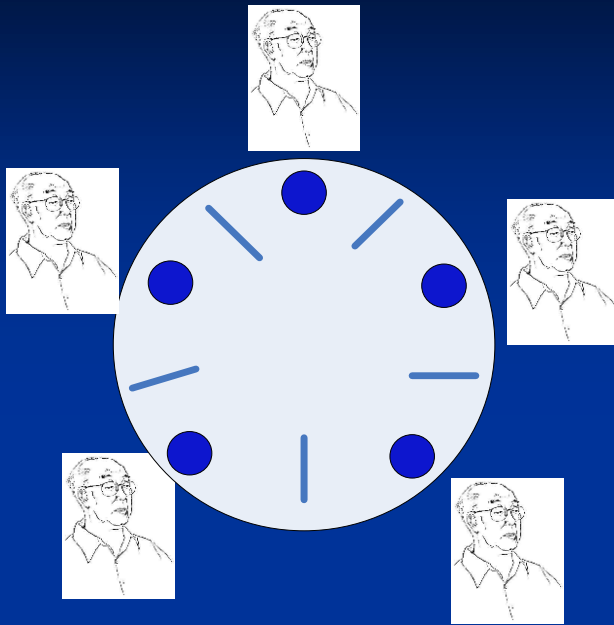
2.4.3 哲学家进餐问题

(1) 哲学家进餐问题

5位哲学家围绕圆桌而坐，反复思考和进餐。但是只有5只碗和筷子，放置如图所示，只有当哲学家同时拿起碗边的2只筷子时，才能进餐。请用记录型信号量进行同步。

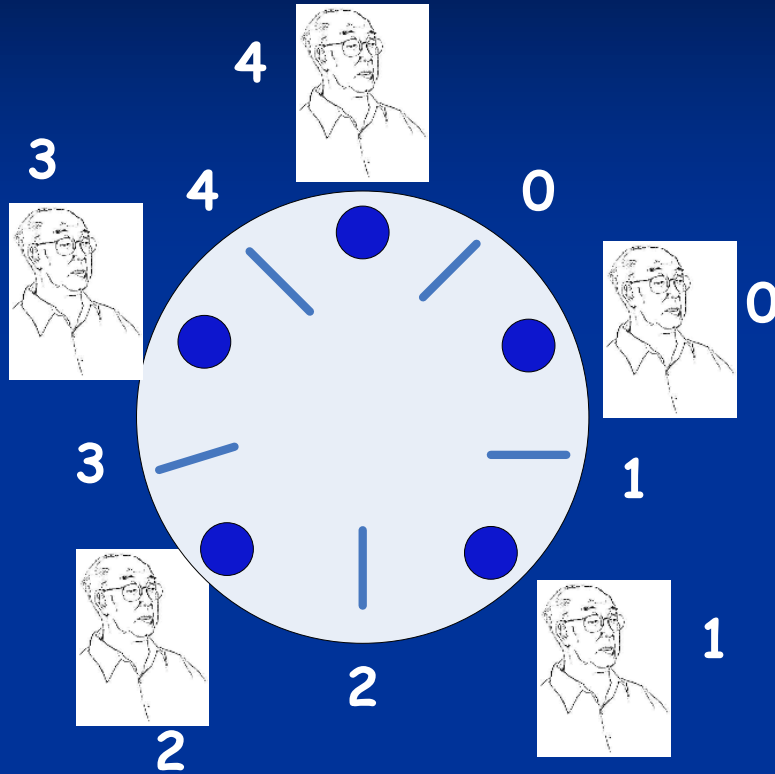


■ 进程资源共享关系和同步关系分析



■ 哲学家进程：互斥（筷子）

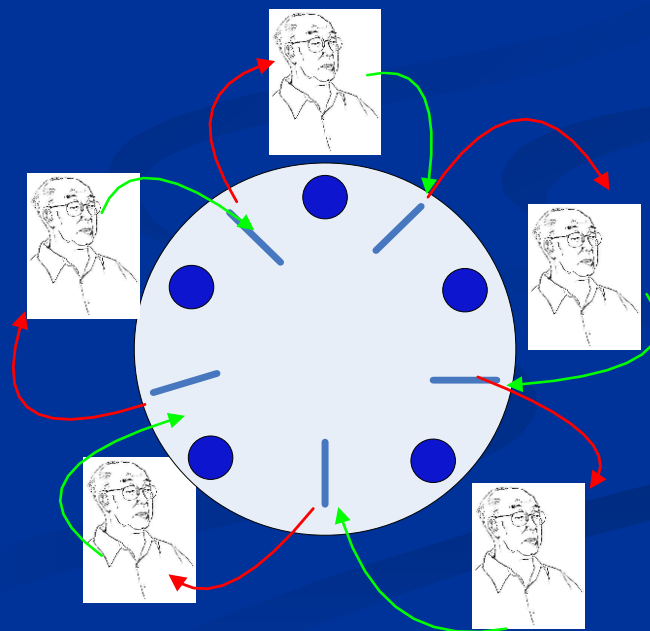
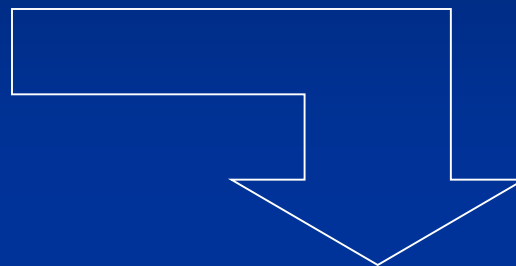
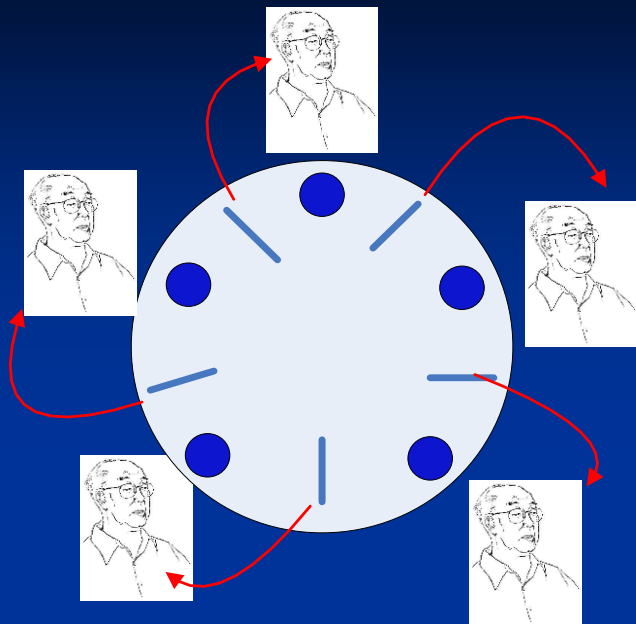
■ 哲学家进餐问题实现



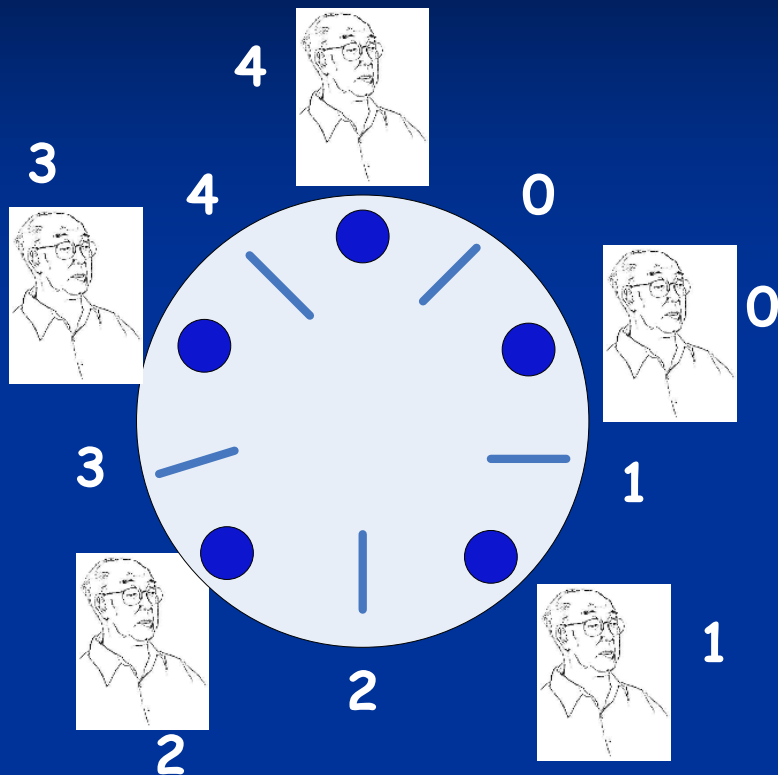
```
VAR chopsticks: array [0..4] of semaphore := 1,1,1,1,1;  
Parbegin  
  philosopheri:  
    begin  
      repeat  
        wait( chopsticks[ i ] );  
        wait( chopsticks[ (i+1) mod 5 ] );  
        eat ;  
        signal( chopsticks[ i ] );  
        signal( chopsticks[ (i+1) mod 5 ] );  
        think  
      until false  
    end  
  Parend
```

■ 问题：正确吗？

(2) 哲学家进餐程序的问题



■ 哲学家进餐标准程序



```

VAR chopsticks: array [0..4] of semaphore := 1,1,1,1,1;
Parbegin
  philosopher0:
  begin
    repeat
      wait( chopsticks[ i ] );
      wait( chopsticks[ (i+1) mod 5 ] );
      eat ;
      signal( chopsticks[ i ] );
      signal( chopsticks[ (i+1) mod 5 ] );
      think
    until false
  end
  philosopher1:
  begin
    repeat
      wait( chopsticks[ (i+1) mod 5 ] );
      wait( chopsticks[ i ] );
      eat ;
      signal( chopsticks[ i ] );
      signal( chopsticks[ (i+1) mod 5 ] );
      think
    until false
  end
end
Parend
  
```

2.4 经典进程同步问题

2.4.4 理发师问题(课后思考题)

(1) 睡眠理发师问题

理发店有一位理发师，一把理发椅和 n 把用来等候理发的椅子。如果没有顾客，则理发师便在理发椅上睡觉，顾客到来时，如理发师闲则理发，否则如有空椅则坐等，没有空椅则离开，编写程序实现理发师和顾客程序，实现进程控制。



2.4 经典进程同步问题

2.4.5 信号量集机制

(1) AND型信号量集机制

■ 基本思想:

- 将进程在整个运行过程中需要的所有资源，**一次性全部地分配**给进程，待进程使用完后再一起释放。
- 对若干个临界资源的分配，采取原子操作方式要么全部分配到进程，要么一个也不分配。
- 为此，在wait操作中，增加了一个“AND”条件，故称为AND同步。

2.4.5 信号量集机制

(1) AND型信号量集机制

■ 实现:

```
type semaphore = record
    value: integer ;           // 资源数量
    L: list of process ;      // 阻塞进程队列
end

function Swait ( var  $S_1, S_2, \dots, S_n$  : Semaphore )
Begin
    if (  $S_1 \geq 1$  and ... and  $S_n \geq 1$  )
        for i := 1 to n do
             $S_i := S_i - 1$  ;
        else
            Place the process in the queue associated with the first  $S_i$  found with  $S_i < 1$ , and
            goto the beginning of Swait.
        end
    function Ssignal(var  $S_1, S_2, \dots, S_n$  : Semaphore )
    Begin
        for i:=1 to n do
             $S_i := S_i + 1$ 
            Remove all process waiting in the queue associated with  $S_i$  into the ready queue.
        end
```

2.4.5 信号量集机制

(1) AND型信号量集机制

■ 应用示例1: 生产者消费者问题

```
VAR mutex, empty, full: semaphore := 1, n, 0 ;  
    in,out: integer := 0, 0 ;  
    Buffer: array [0..n-1] of item ;  
Parbegin  
    Producer:  
    begin  
        repeat  
            produce an item in nextp ;  
            Swait( empty, mutex ) ;  
            Buffer(in) := nextp ;  
            in := (in + 1) mod n ;  
            Ssignal( mutex, full ) ;  
        until false  
    end  
    Consumer:  
    begin  
        repeat  
            Swait( full,mutex ) ;  
            nextc = Buffer(out);  
            out := (out + 1) mod n ;  
            Ssignal( mutex,empty ) ;  
            consume the item nextc ;  
        until false  
    end  
Parend
```

2.4.5 信号量集机制

(1) AND型信号量集机制

■ 应用示例2：哲学家进餐问题

```
VAR chopsticks: array [0..4] of semaphore := 1,1,1,1,1 ;
Parbegin
  philosopheri:
  begin
    repeat
      Swait( chopsticks[ i ], chopsticks[ (i+1) mod 5 ] ) ;
      eat ;
      Ssignal( chopsticks[ i ], chopsticks[ (i+1) mod 5 ] ) ;
      think
    until false
  end
Parend
```

2.4 经典进程同步问题

2.4.5 信号量集机制

(2) 一般信号量集机制

■ 基本思想:

□ 在AND型信号量基础上，一次可申请多个单位资源。

2.4.5 信号量集机制

(2) 一般信号量集机制

■ 实现:

```
type semaphore = record
    value: integer ;           // 资源数量
    L: list of process ;      // 阻塞进程队列
end

function Swait ( var  $S_1, t_1, d_1, S_2, t_2, d_2, \dots, S_n, t_n, d_n$  : Semaphore )
Begin
    if (  $S_1 \geq t_1$  and ... and  $S_n \geq t_n$  )
        for i := 1 to n do
             $S_i := S_i - d_i ;$ 
        else
            Place the process in the queue associated with the first  $S_i$  found with  $S_i < t_i$ , and
            goto the beginning of Swait.
        end
    function Ssignal(var  $S_1, d_1, S_2, d_2, \dots, S_n, d_n$  : Semaphore )
    Begin
        for i:=1 to n do
             $S_i := S_i + d_i$ 
            Remove all process waiting in the queue associated with  $S_i$  into the ready queue.
        end
```

2.4.5 信号量集机制

(2) 一般信号量集机制

■ 应用实例：读者写者问题

RN: 允许读者进程上限

```
VAR RN: integer ;  
    L,mutex: semaphore := RN, 1 ;  
Parbegin  
  Reader:  
  begin  
    repeat  
      Swait( L,1,1, mutex,1,0 ) ;  
      perform read operation ;  
      Ssignal( L, 1 ) ;  
    until false  
  end  
  Writer:  
  begin  
    repeat  
      Swait( mutex,1,1, L,RN,0 ) ;  
      perform write operation;  
      Ssignal( mutex,1 ) ;  
    until false  
  end  
Parend
```

□ 问题分析
可否再简化程序？