



# 计算机操作系统

---

Operating Systems

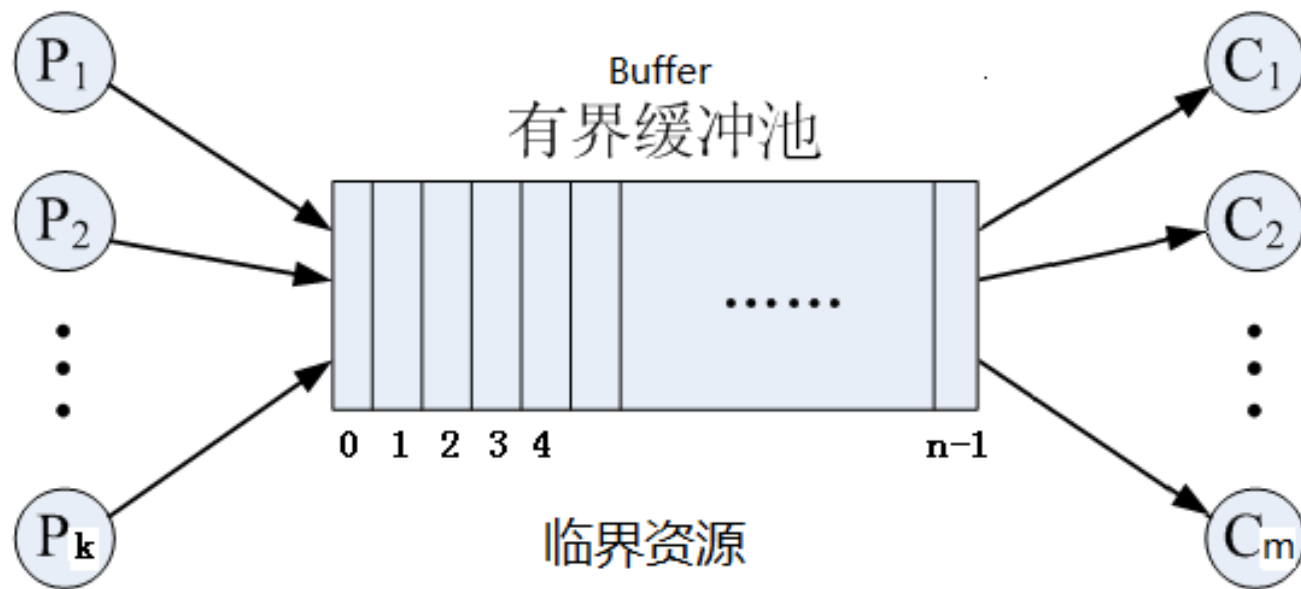
李琳

## 第二章 进程的描述与控制

## 2.5 经典进程同步问题

### • 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。



## 2.5 经典进程同步问题

### • 单人单缓

VAR **empty**, **full**: semaphore := 1, 0 ;  
Buffer : item;

Parbegin

**Producer:**

begin

repeat

produce an item in P ;

**wait( empty );**

Buffer = P ;

**signal( full );**

until false

end

**Consumer:**

begin

repeat

**wait( full );**

C = Buffer;

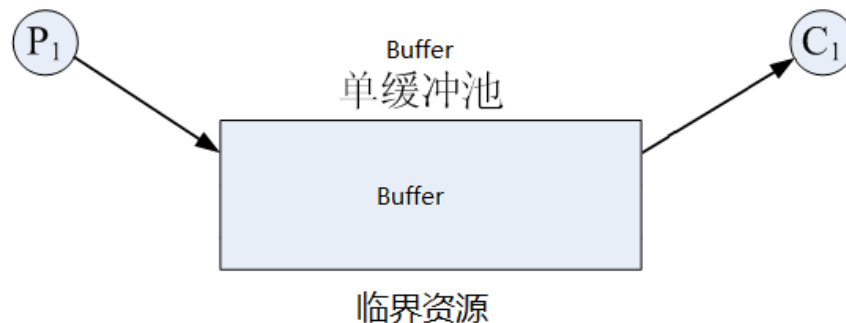
**signal( empty );**

consume the item C ;

until false

end

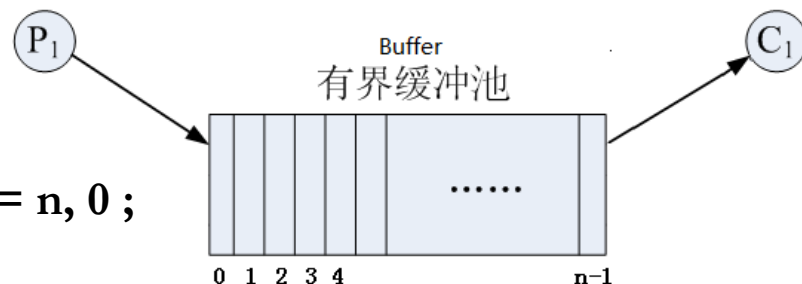
Parend



请思考empty和full的最大最小值？ 各在什么情况下发生？

$-1 \leq \text{Empty} \leq 1$   
 $-1 \leq \text{full} \leq 1$

## 2.5 经典进程同步问题



### • 单人多缓

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 P ;
    wait( empty );
    Buffer(in) = P ;
    in := (in + 1) mod n ;
    signal( full );
  until false
end
```

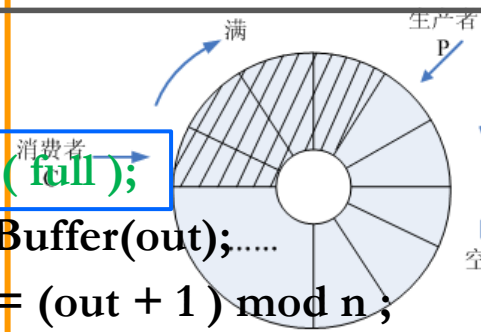
**Consumer:**

```
begin
  repeat
    wait( full );
    C = Buffer(out); ....
    out := (out + 1) mod n ;
    signal( empty )
    consume the item C ;
  until false
end
```

Parend

临界资源

用循环队列描述多缓冲区



请思考empty和full的最大最小值？ 各在什么情况下发生？

$-1 \leq \text{Empty} \leq n$   
 $-1 \leq \text{full} \leq n$

## 2.5 经典进程同步问题

### • 多人单缓

```
VAR empty, full: semaphore := 1, 0 ;  
    Buffer : item;
```

Parbegin

Producer(1-k):

begin

repeat

produce an item in P ;

wait( empty );

Buffer= P ;

signal( full );

until false

end

Consumer(1-m):

begin

repeat

wait( full );

C = Buffer;

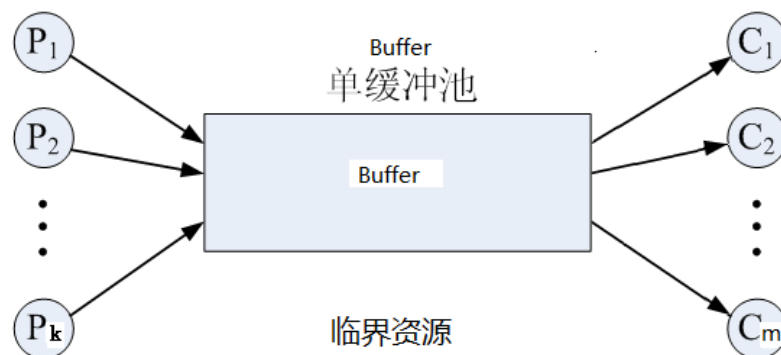
signal( empty );

consume the item C ;

until false

end

Parend



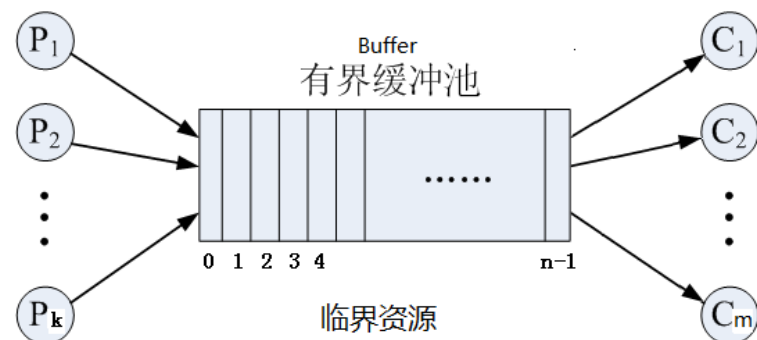
请思考empty和full的最大最小值？ 各在什么情况下发生？

-k ≤ Empty ≤ 1  
-m ≤ full ≤ 1

## 2.5 经典进程同步问题

### • 多人多缓

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



Parbegin

Producer:

Consumer:

```
begin
  repeat
    produce an item in P ;
    wait( empty );
    wait( mutex );
    Buffer(in) = P ;
    in := (in + 1) mod n ;
    signal( mutex );
    signal( full );
  until false
end
```

生产者之间互斥

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

消费者之间互斥

Parend

请思考empty和full的最大最小值？各在什么情况下发生？mutex呢？

$-k \leq \text{Empty} \leq n$   
 $-m \leq \text{full} \leq n$

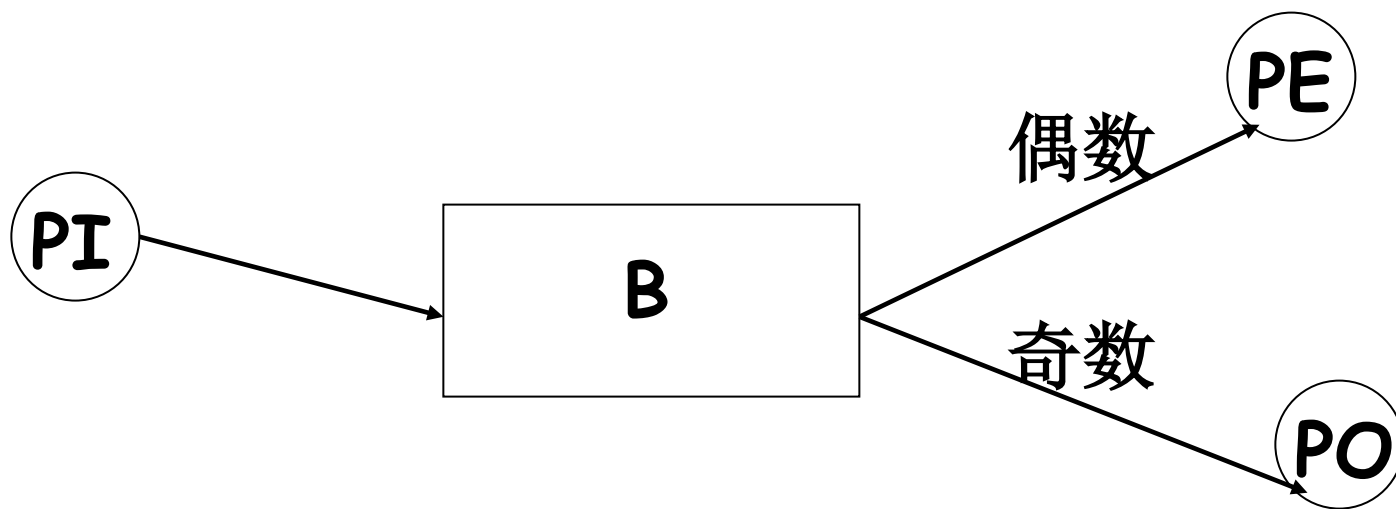
# 生产者消费者问题思考

- (1) 允许生产者写时，消费者可读
- (2) 当缓冲区无限大；
- (3) 调整生产者wait顺序；
- (4) 调整消费者wait顺序；
- (5) 调整signal顺序



## 试一试

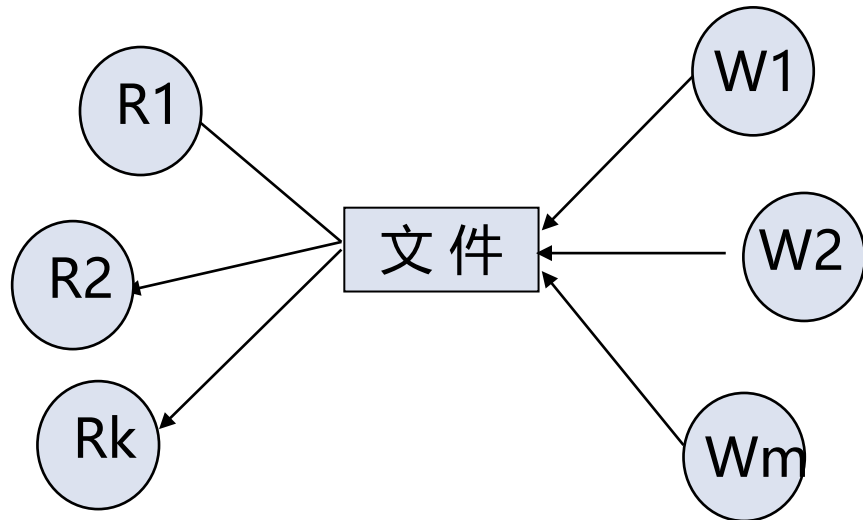
- PI, PE, PO三进程通过共享缓冲区B协作, PI负责产生随机数并保存在B中, 当B中数据为奇数时由PO负责打印, 当B中数据为偶数时由PE负责打印, 要求每个数据必须且只打印一次, 不许漏打或重复打印。请用信号量进行同步。



## 2.5 经典进程同步问题

### • 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。



同时读

分开写

- 读者进程：有条件的互斥
- 写者进程：无条件的互斥
- 读者写者顺序：无

## 2.5 经典进程同步问题

```
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
```

当有读者在访问时，不需要申请互斥资源 mutex

最后一个读者离开时，才需要通知写者

```
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
```

时间片到?

- 有什么问题?

## 2.5 经典进程同步问题

VAR **rmutex**, **wmutex**: semaphore := 1, 1;  
    **readcount**: integer := 0;

- **rmutex** 是多个读者进程对变量 **readcount** 进行互斥的信号量
- **wmutex** 是读者进程有条件的与写者进程对文件进行互斥的信号量
- 请思考 **rmutex** 和 **wmutex** 的最大最小值？各在什么情况下发生？

$$\begin{aligned} -m &\leq wmutex \leq 1 \\ -k+1 &\leq rmutex \leq 1 \end{aligned}$$

```
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
```

临界区

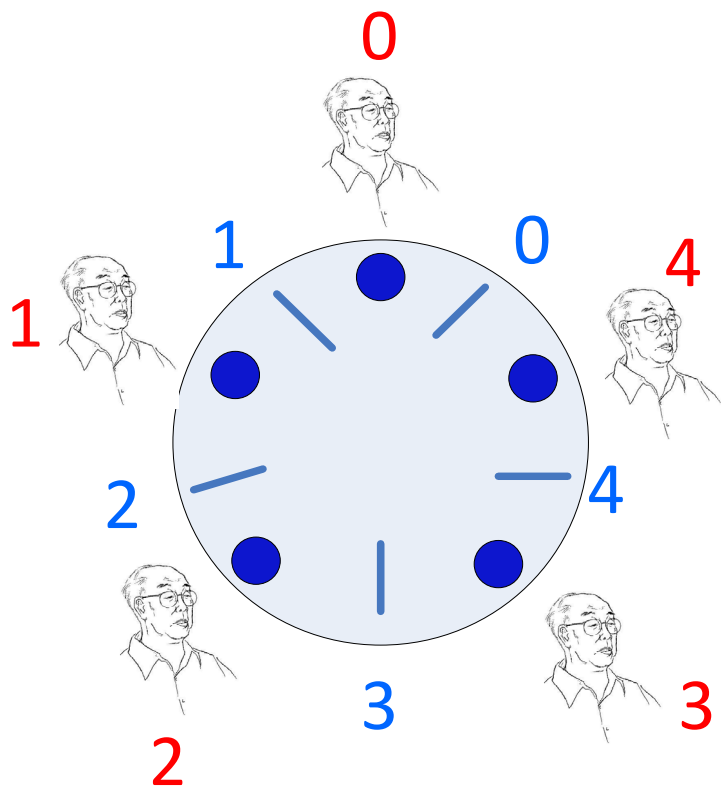
临界区

临界区

## 2.5 经典进程同步问题

### • 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。



```
VAR chopsticks: array [0..4] of  
semaphore := 1,1,1,1,1;
```

```
Parbegin
```

```
philosopher i:
```

```
begin
```

```
repeat
```

```
wait( chopsticks[ i ] );
```

```
wait( chopsticks[ (i+1) mod 5 ] );
```

```
eat ;
```

```
signal( chopsticks[ i ] );
```

```
signal( chopsticks[ (i+1) mod 5 ] );
```

```
think
```

```
until false
```

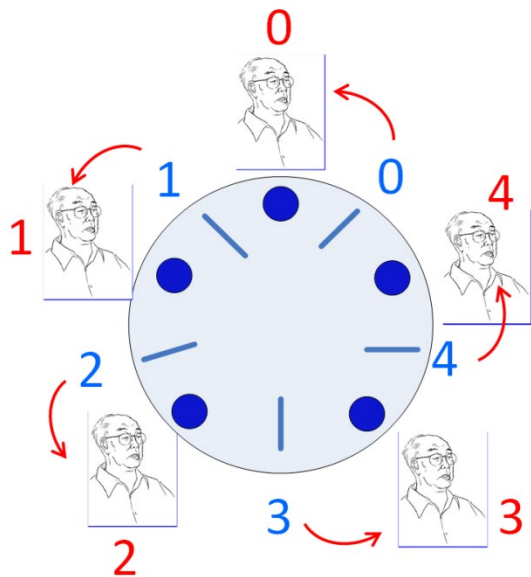
```
end
```

```
Parend
```

时间片到?

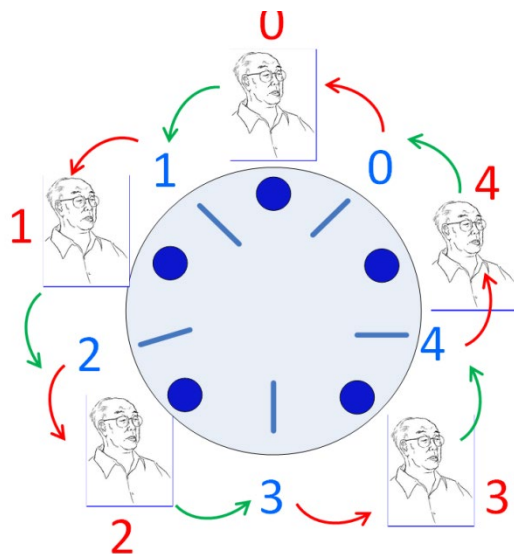
• 有没有问题?

## 2.5 经典进程同步问题



都申请到了左筷子

有人改变申请次序



都去申请右筷子，  
形成环路

VAR chopsticks: array [0..4] of semaphore :=  
1,1,1,1,1;

Parbegin

philosopher0-3:

begin

repeat

wait( chopsticks[ i ] );

wait( chopsticks[ (i+1) mod 5 ] );

eat ;

signal( chopsticks[ i ] );

signal( chopsticks[ (i+1) mod 5 ] );

think

until flase

end

philosopher4:

begin

repeat

wait( chopsticks[ (i+1) mod 5 ] );

wait( chopsticks[ i ] );

eat ;

signal( chopsticks[ i ] );

signal( chopsticks[ (i+1) mod 5 ] );

think

until flase

end


Parend

# 信号量的其它表示方法

- AND型信号量
- 信号量集

wait(s1) **AND** wait(s2) .....  
if s1>=1            if s2>=1  
    s1=s1-1          s2=s2-1



Swait(s1,s2)  
if (s1>=1)&&(s2 >=1)      
    s1=s1-1; s2=s2-1;

Swait(s1,t1,d1,s2,t2,d2).....  
if (s1>=t1)&&(s2 >=t2)  
    s1=s1-d1; s2=s2-d2;

Windows下所提供

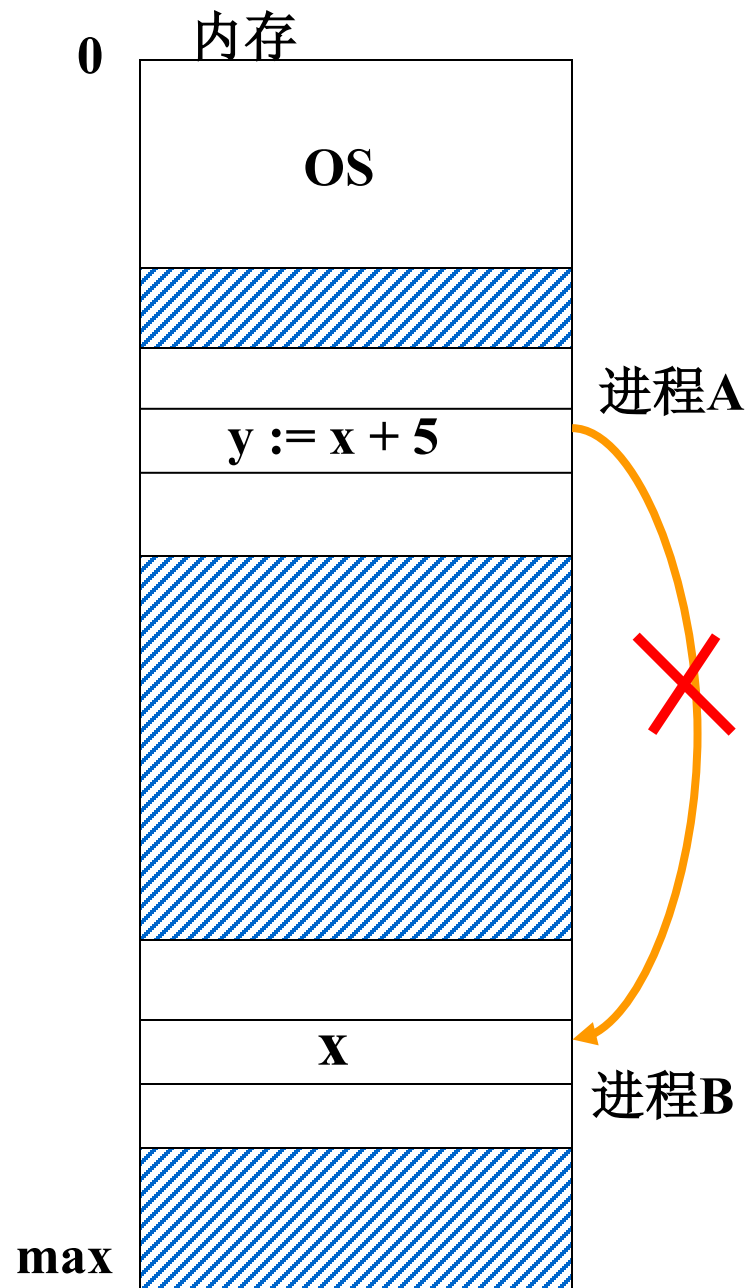
DWORD [WaitForSingleObject](#) (HANDLE  
hHandle, DWORD dwMilliseconds );

DWORD [WaitForMultipleObjects](#)(DWORD  
nCount, const HANDLE\* lpHandles, BOOL  
bWaitAll, DWORD dwMilliseconds );

## 2.6 进程通信

- 进程间通信 ( Inter Process Communication, IPC )

- ✓ 进程通信指进程之间的信息交换，其所交换的信息量，少者是一个状态或数值，多者则是成千上万个字节。
- ✓ 进程是不能访问另外一个进程的内存空间的
- ✓ 信号量方法可以在两个进程间传送一个数值





## 2.6 进程通信

### 2.6.1 共享存储器系统

- 基本思想

- ✓ 相互通信的进程共享数据结构或共享存储区，进程之间通过这些空间进行通信。

- 实现过程

- ① 进程申请一片内存，拿到内存描述符；
- ② 如果已分配过，直接获得内存描述符；
- ③ 利用内存描述符将内存连接到本进程；
- ④ 读写

- 访问同步问题

- 示例：linux共享内存、剪贴板

# Linux下的共享内存

①输入IPC标志

`int shmget(key_t key, int size, int flag);`

②标识共享内存

`void *shmat(int shmid, void *addr, int flag);`

③合并地址空间

④使用共享内存

`int shmctl(int shmid, int cmd, shmid_ds *buf);`

操纵共享内存

↓  
设置、获取、删除

↓  
共享内存信息

## 2.6 进程通信

### 2.6.2 消息通信系统

- 基本思想

- ✓ 进程间的数据交换，是以格式化的消息(message)为单位进行通信。

- 直接通信——同步

- ✓ 信息直接传递给接收方

- ✓ 发送进程: Send(receiver,msg);

- ✓ 接收进程: Receive (sender,msg);

- 间接通信——异步

- ✓ 使用一个中间存储区进行消息传递

- ✓ 发送进程: Send(mailbox,msg);

- ✓ 接收进程: Receive(mailbox,msg);

消息的基本结构

```
type message buffer=record
```

```
    sender ; 发送者进程标识符
```

```
    size   ; 消息长度
```

```
    text   ; 消息正文
```

```
    next   ; 指向下一个消息缓冲区的指针
```

```
end
```

## Windows的消息循环

```
while(GetMessage(&&msg,NULL,NUL  
L,NULL))  
{    //从消息队列中取得消息  
    TranslateMessage(&&msg);  
    //检索并生成字符消息WM_CHAR  
    DispatchMessage(&&msg);  
    //将消息发送给相应的窗口函数  
}
```

SendMessage(hwnd,msg,wParam,lParam)    **同步方式**

PostMessage(hwnd,msg,wParam,lParam)    **异步方式**

## Linux下的消息缓冲

int msgget(key\_t key, int flag); //创建或  
打开一个消息队列

int msgsnd(int msqid, const void \*ptr,  
size\_t nbytes, int flag); //发送一条消息

ssize\_t msgrcv(int msqid, void \*ptr,  
size\_t nbytes, long type, int flag); //取出  
一条消息

## 2.6 进程通信

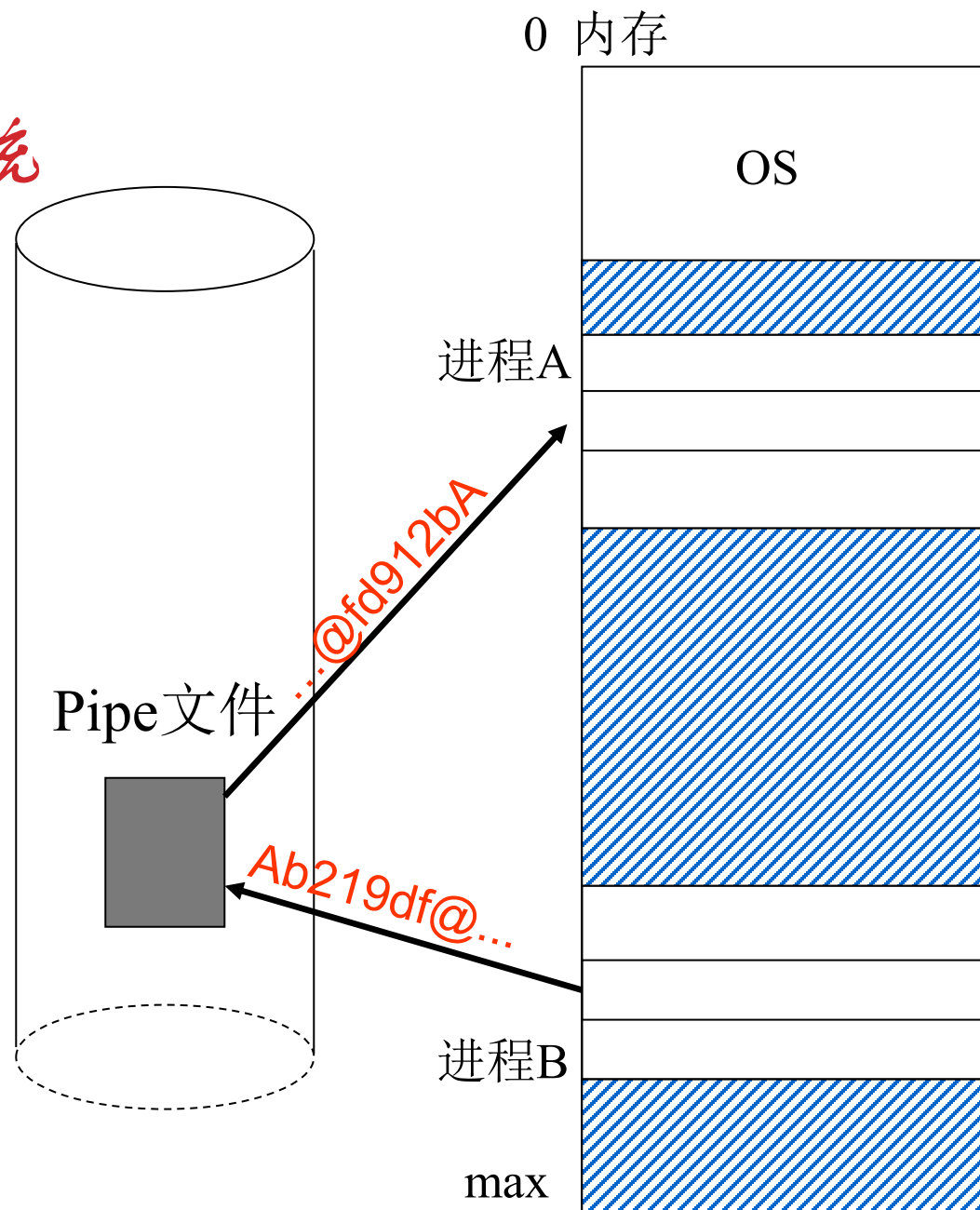
### 2.6.3 管道通信系统

#### • 基本思想

- ✓ 在磁盘创建一个文件，大小固定，如4kB
- ✓ 发送进程写连续字符流
- ✓ 接收进程按顺序读连续字符流
- ✓ 管道将一个进程的输出和另一个进程的输入链接在一起

#### • 实现技术

- ✓ 对管道文件的共享：同步、互斥
- ✓ UNIX首创



# Windows的管道通信

- Windows提供**无名管道**和**命名管道**两种管道机制。
- 利用CreatePipe可创建无名管道并得到两个读写句柄；
- 利用ReadFile和WriteFile可并行无名管道的读写。

```
BOOL CreatePipe(PHANDLE  
hReadPipe, // 指向读句柄的指针  
                PHANDLE hWritePipe, // 指向写句柄的指针  
                LPSECURITY_ATTRIBUTES  
lpPipeAttributes, // 指向安全属性的指针  
                DWORD nSize // 管道大小  
);
```

# Linux的管道通信

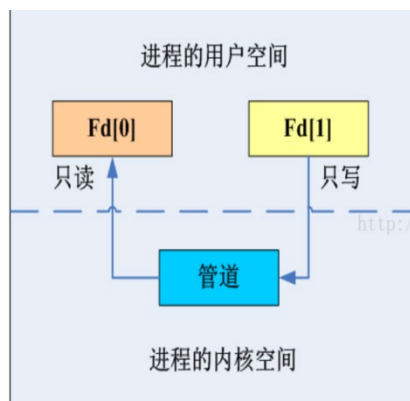
所需头文件 `#include <unistd.h>`

函数原型 `int pipe(int fd[2])`

函数传入值 **fd[2]**: 用于保存管道的两个文件描述符，之后就可以直接操作这两个文件描述符

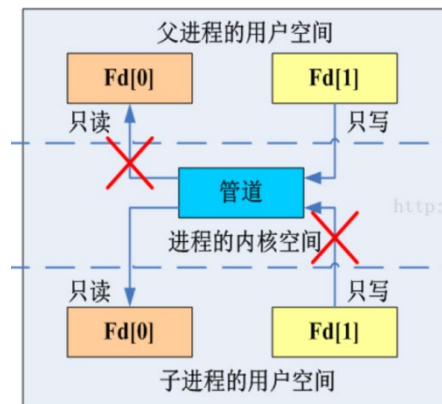
函数返回值 成功: **0**

出错: **-1**



当进程创建一个管道之后的连接情况

父进程写、子进程读



当确定连接关系之后管道的连接情况

## 2.6 进程通信

### Windows的IPC方法

- 内存文件映像
- 管道
- 剪贴板
- 动态数据交换(DDE)
- WM\_COPYDATA消息
- Sockets

### Linux的IPC方法

- 信号量
- 共享内存
- 消息队列
- 管道

## 第二次作业

1、为了喝水，口渴的人必须有三件物品：水、冰和杯子。有三个口渴的人，每人有且只有这三个必需品里不同的一个。第4个人是服务员，可以无限制的提供三个必需品。如果没有人在喝水，服务员将三件物品中的任意两个（随机选取）放在桌子上。能够用这两个物品喝水的人将拿起它们并喝一杯冰水。喝完后，口渴的人将通知服务员，并重复这一过程。试写出4个人的并发程序，并设计信号量用PV操作限制其同步过程。

2、伊拉克战争期间，美步3师攻占了底格里斯河上的一座桥梁。底格里斯河将伊拉克首都巴格达分为东、西两部分。美军的油料基地设置在河西岸，不断有东岸的美军坦克经过该桥到西岸去加油，也不断有西岸的坦克通过该桥到东岸投入战斗。由于这是一座旧桥，桥面只比坦克略宽，不允许两辆坦克并排行使；桥的载重也有限，最多允许5辆坦克同时在桥上开行。请为美军宪兵设计一个调度系统，使用Wait/Signal操作（P/V操作）协调两岸坦克对桥的使用。

3、学习WINDOWS的线程编程API，并编写三大经典问题之一，自己设计线程个数、资源情况，演示线程运行过程，不需要窗口可视化，使用控制台程序。（打印程序和输出结果贴在作业本上）