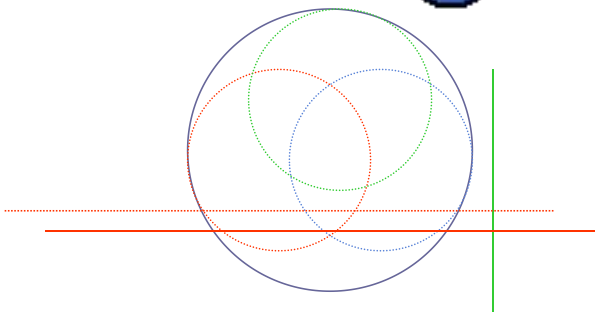
操作系统
第十讲

张涛

Review

信号量和PV原语操作

经典进程同步问题



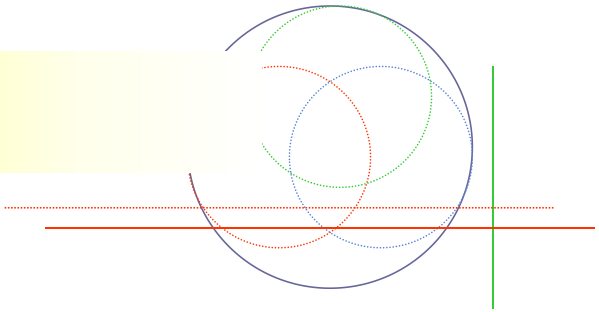
3.6.3 进程间通信
IPC, INTER-PROCESS COMMUNICATION

进程间通信类型

消息缓冲通信的实现

信箱通信的实现

管道通信的实现



基本概念

- 进程通信是指进程之间可直接以较高的效率传递较多数据的信息交换方式。
- 低级通信：只能传递状态和控制信息等，进行简单的信号交换。
  - 优点：速度快
  - 缺点：
    - 传送信息量小：效率低，每次通信传递的信息量固定，若传递较多信息则需要进行多次通信。
    - 编程复杂：用户直接实现通信的细节，易出错。
- 高级通信：高效、大量的数据传递。

高级通信的特征

- 通信链路(communication link)：
  - 点对点/多点/广播
  - 单向/双向
  - 有容量（链路带缓冲区）/无容量（发送方和接收方需自备缓冲区）
- 数据格式：
  - 字节流(byte stream)：各次发送之间的分界，在接收时不被保留，没有格式；
  - 报文(datagram/message)：各次发送之间的分界，在接收时被保留，通常有格式，定长/不定长报文，可靠/不可靠报文。
- 收发操作的同步方式
  - 发送阻塞和不阻塞
  - 接收阻塞和不阻塞
  - 由事件驱动收发：在允许发送或有数据可读时，才做发送和接收操作

3.6.3.1 进程通信类型

- 共享存储器系统
  - 通过数据、数据区的共享，写入与读出达到通信的目的
- 消息传递系统
  - 直接通信方式：消息缓冲
    - 采用进程的消息缓冲队列
    - 消息发送者将消息直接放在接收者的消息缓冲队列
  - 间接通信方式：邮箱通信
    - 利用中间者——信箱、邮局来传递信件。
    - 发送进程将消息发送到信箱中，接收进程从信箱中取出消息
- 管道通信 (共享文件方式)
  - 用以连接读、写进程的共享文件

1. 共享存储器系统
Shared-Memory System

- 基于共享数据结构的通信方式
  - 诸进程公用某些数据结构，进程通过它们交换信息。如生产者-消费者问题中的有界缓冲区。
- 基于共享存储区的通信方式
  - 高级通信，在存储器中划出一块共享存储区，进程在通信前，向系统申请共享存储区中的一个分区，并指定该分区的关键字，若系统已经给其它进程分配了这样的分区，则将该分区的描述符返回给申请者。接着，申请者把获得的共享存储分区连接到本进程上，此后可读写该分区。

以上两种方式的同步互斥都要由进程自己负责。

UNIX的共享存储区

- 创建或打开共享存储区(shmget)：依据用户给出的整数值key，创建新区或打开现有区，返回一个共享存储区ID。
- 连接共享存储区(shmat)：连接共享存储区到本进程的地址空间，可以指定虚拟地址或由系统分配，返回共享存储区首地址。父进程已连接的共享存储区可被fork创建的子进程继承。
- 拆除共享存储区连接(shmdt)：拆除共享存储区与本进程地址空间的连接。
- 共享存储区控制(shmctl)：对共享存储区进行控制。如：共享存储区的删除需要显式调用shmctl(shmid, IPC_RMID, 0);

2. 消息传递系统
Message passing system

- 进程间的数据交换以消息为单位，程序员利用系统的通信原语实现通信。
- 消息传递系统可分为：
  - 直接通信：发送进程直接把消息发送给接收者，并将它挂在接收进程的消息缓冲队列上。接收进程从消息缓冲队列中取得消息。也称为消息缓冲通信
  - 间接通信：发送进程将消息发送到某种中间实体中（信箱），接收进程从中取得消息。也称信箱通信。在网络中称为电子邮件系统。
- 两种方式的主要区别？
  - 前者需要两进程都存在，后者不需要。

3. 管道通信 Pipe

- 所谓管道，是指用于连接一个读进程和一个写进程的文件，称pipe文件。向管道提供输入的进程（称写进程），以字符流的形式将大量数据送入管道，而接受管道输出的进程（读进程）可从管道中接收数据。该方式首创于UNIX，它能传送大量数据，被广泛采用。

字符流方式写入读出
先进先出顺序

3.6.3.2 消息缓冲通信的实现

- 用消息传递的方式(传递数据块)达到通信的目的

  1)系统管理空白缓冲池

  2)发送者向系统申请空白缓冲区

  3)发送者将填有消息的缓冲区挂到接收者的消息队列

  4)接收者在适当时候从消息队列中读取数据

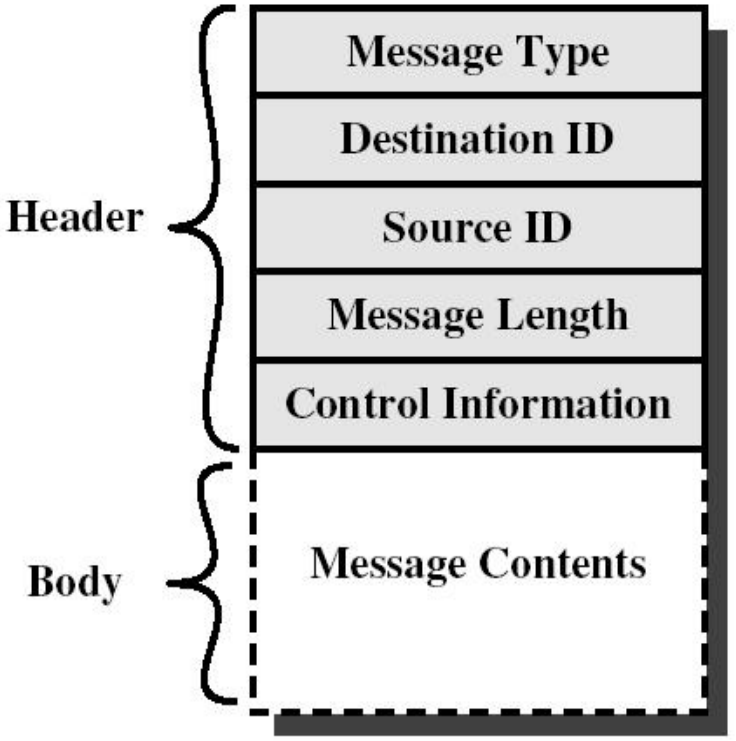
  5)系统提供消息通信原语：

  - send_message( ) ,
  - receive_message( );

每个接收者有自己的消息队列

消息

- 消息：一段文本。消息格式设计与应用环境和要求有关
  - 固定长度消息：可以减小处理和存储的开销
  - 基于文件的：传送大量的数据
  - 可变长度消息：灵活
- 消息的一般格式
  - 消息头：源标识、目的标识、
    长度域、类型域、控制域
  - 消息体



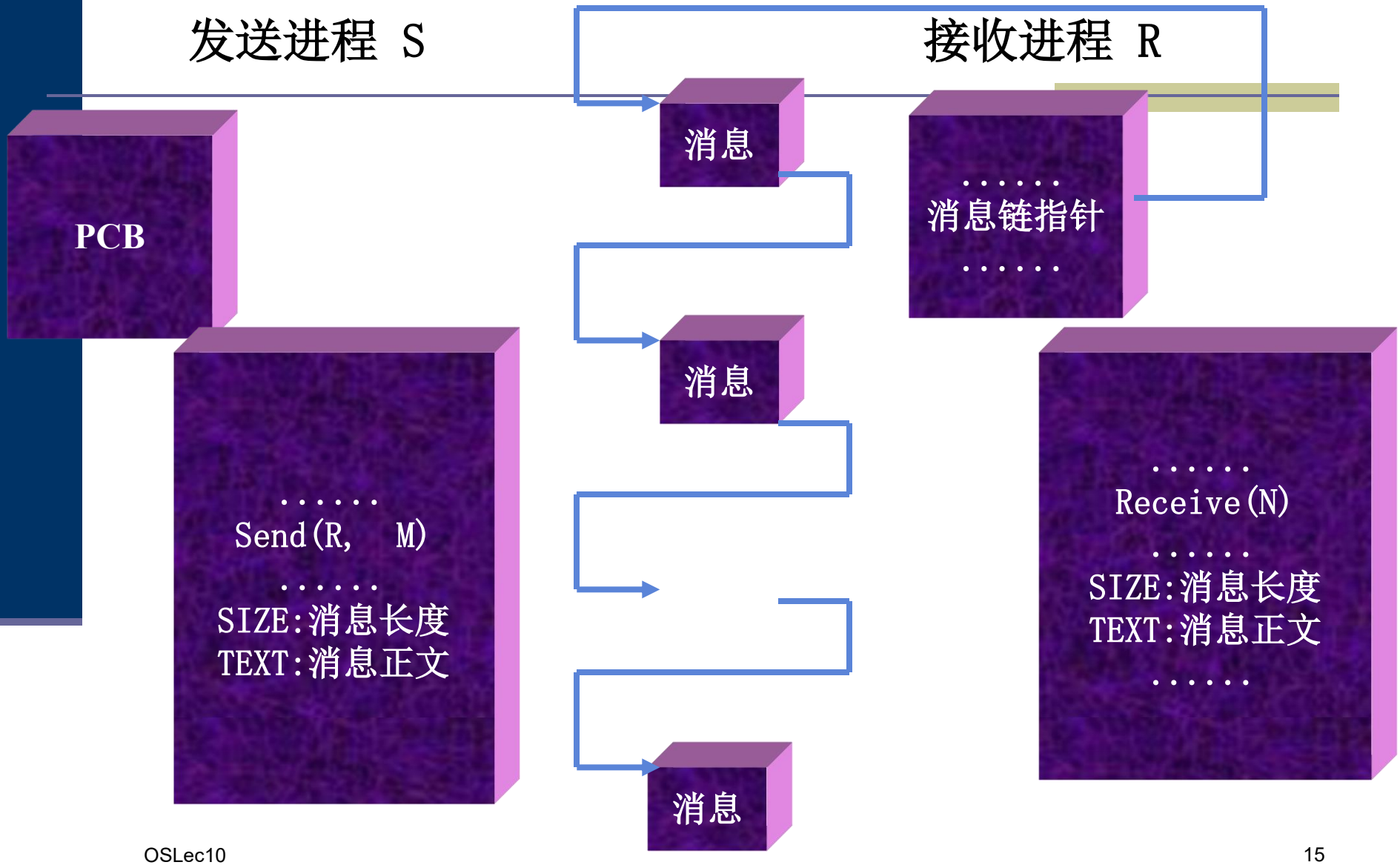
消息缓冲区结构

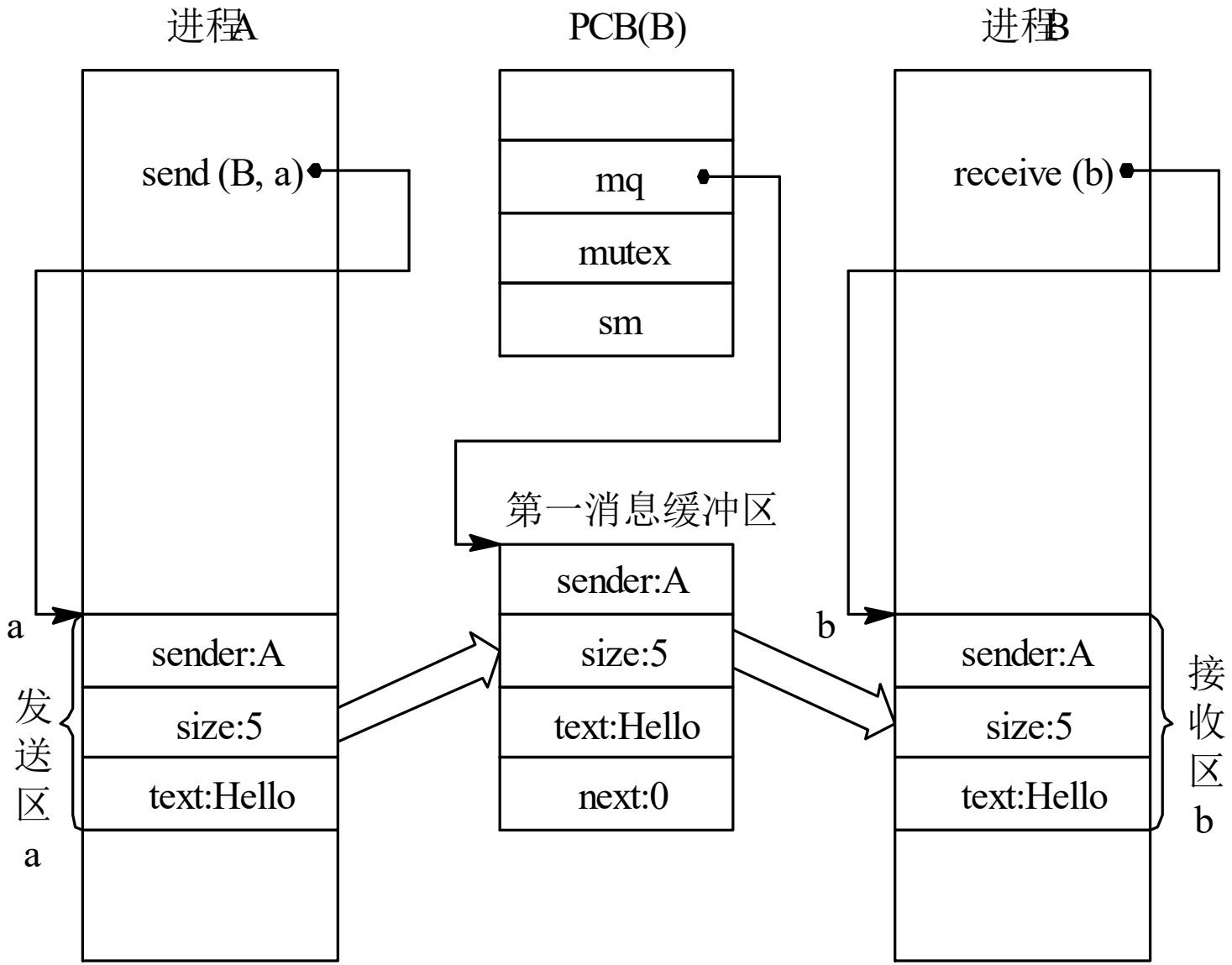
```
type message Buffer=record
  sender ;        //发送者ID
  size   ;        //消息长度
  text   ;        //消息正文
  next   ;        //消息队列指针
end
```

PCB中有关通信的数据项

```
type processcontrol block=record
    ...
    mq;          //消息队列队首指针
    mutex;       //消息队列互斥信号量
    sm; //消息队列资源信号量
    ...
end
```

当进程向一个满队列发送消息时，它将被挂起；
当进程从一个空队列读取时也会被挂起。


发送进程 S
接收进程 R
PCB
消息
消息
消息
.....
消息链指针
.....
.....
Send(R, M)
.....
SIZE:消息长度
TEXT:消息正文
.....
Receive(N)
.....
SIZE:消息长度
TEXT:消息正文
.....


进程A
PCB(B)
进程B
send (B, a)
mq
mutex
sm
receive (b)
第一消息缓冲区
sender:A
size:5
text:Hello
next:0
a
发送区a
sender:A
size:5
text:Hello
b
sender:A
size:5
text:Hello
接收区b

消息缓冲通信

用P、V操作来实现Send原语

```
send（R,M）
begin
  在OS中分配M.size大小的缓冲区t;
  将M中的内容复制到t;
  得到进程R的PCB的指针q;
  P(q.mutex);
        将t挂到队列q.mq队尾;
  V(q.mutex);
  V（q.sm);
end
```

用P、V操作来实现Receive原语

```
Receive(N)
begin
  得到本进程PCB的指针q;
  P(q.sm);
  P(q.mutex);
    从q.mq队首取下一个缓冲区t;
  V(q.mutex);
  将t的内容复制到N, 并释放t
end
```

```
procedure send(receiver, a)
    begin
      getbuf(a.size,i);                          //根据a.size申请缓冲区；
      i.sender：= a.sender;   // 将发送区a中的信息复制到缓冲区;
      i.size：   = a.size;
      i.text：   =  a.text;
      i.next：   = 0;
     getid(PCB set, receiver.j);        // 获得接收进程内部标识符；
     wait(j.mutex);
     insert(j.mq, i);                          // 将消息缓冲区插入消息队列；
     signal(j.mutex);
     signal(j.sm);
end
```

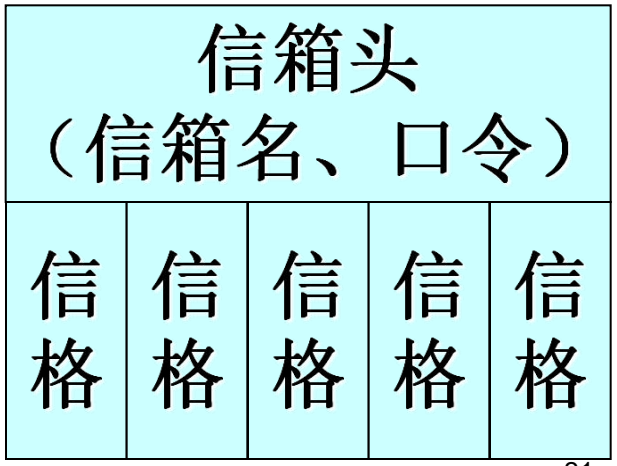
```
procedure receive(b)
  begin
   j:    =internal name;          //j为接收进程内部的标识符;
   wait(j.sm);
   wait(j.mutex);
   remove(j.mq, i);               //将消息队列中第一个消息移出;
   signal(j.mutex);
   b.sender:    =i.sender;        //将缓冲区i中的信息复制到接收区b;
   b.size:     =i.size;
   b.text:     =i.text;
 end
```

3.6.3.3 信箱通信的实现

- 信箱的通讯过程

- 信箱的数据结构

```
type mailbox Buffer=record
    boxname ;          //信箱名
    boxsize  ;//信箱大小
    mesnum  ;          //已存信件数
    fromnum ;          //空的格子数
    。 。 。 。 。 。
 end
```



信箱的使用

- 信箱的创建和撤消
- 消息的发送和接收

Send(mailbox，message)：将一个消息发送到指定信箱；
Receive(mailbox，message)：从指定信箱中接收一个消息；

- 信箱使用规则
  - 若发送信件时信箱已满，则发送进程被置为“等信箱”状态，直到信箱有空时才被唤醒
  - 若取信件时信箱中无信，则接收进程被置为“等信件”状态，直到有信件时才被唤醒

Send实现

- send（MailBox，M）：把信件M送到指定的信箱MailBox中
- 步骤：
  - 查找指定信箱MailBox ；
  - 若信箱未满，则把信件M送入信箱且唤醒“等信件”者；
  - 若信箱已满置发送信件进程为“等信箱”状态；

Receive实现

- receive（MailBox，X）：从指定信箱MailBox中取出一封信，存放到指定的地址X中
- 步骤：
  - 查找指定信箱MailBox；
  - 若信箱中有信，则取出一封信存于X中且唤醒“等信箱”者；
  - 若信箱中无信件则置接收信件进程“等信件”状态；

信箱的分类

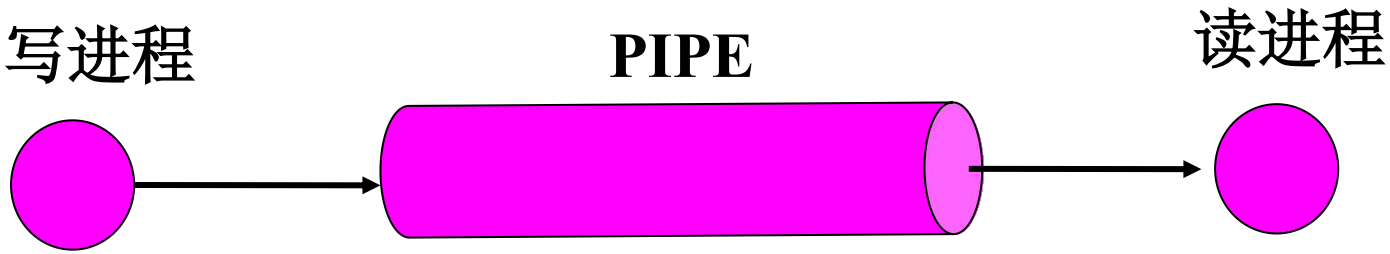
- 私用信箱
  - 用户进程可为自己建立一个新信箱，并作为该进程的一部分。信箱的拥有者有权从信箱中读取消息，其他用户则只能将自己构成的消息发送到该信箱中。
- 公用信箱
  - 由操作系统创建，并提供给系统中的所有核准进程使用。核准进程既可把消息发送到该信箱中，也可从信箱中读取发送给自己的消息。
- 共享信箱
  - 由某进程创建，在创建时或创建后，指明它是可共享的，同时须指出共享进程(用户)的名字。信箱的拥有者和共享者，都有权从信箱中取走发送给自己的消息。

消息传递方式－类型

- 直接通信：必须明确命名通信的接收者或发送者，原语send和receive的定义如下：
  - Send(P,message):发送消息到进程P
  - Receive(Q,message):接收来自进程Q的消息。
- 间接通信：消息通过邮箱或端口来发送和接收。原语定义如下：
  - Send（A,message）；
  - receive（A,message）；

3.6.3.4 管道通信的实现

- 管道：是所有Unix都提供的一种IPC机制
  - 一个进程将数据写入管道，另一个进程从管道中读取数据
  - 是连接接收进程和发送进程的共享文件
    - （1）互斥访问
    - （2）写后读，读后写的同步
    - （3）只有在管道双方都存在时才能通信
  - 无名管道，有名管道



UNIX无名管道

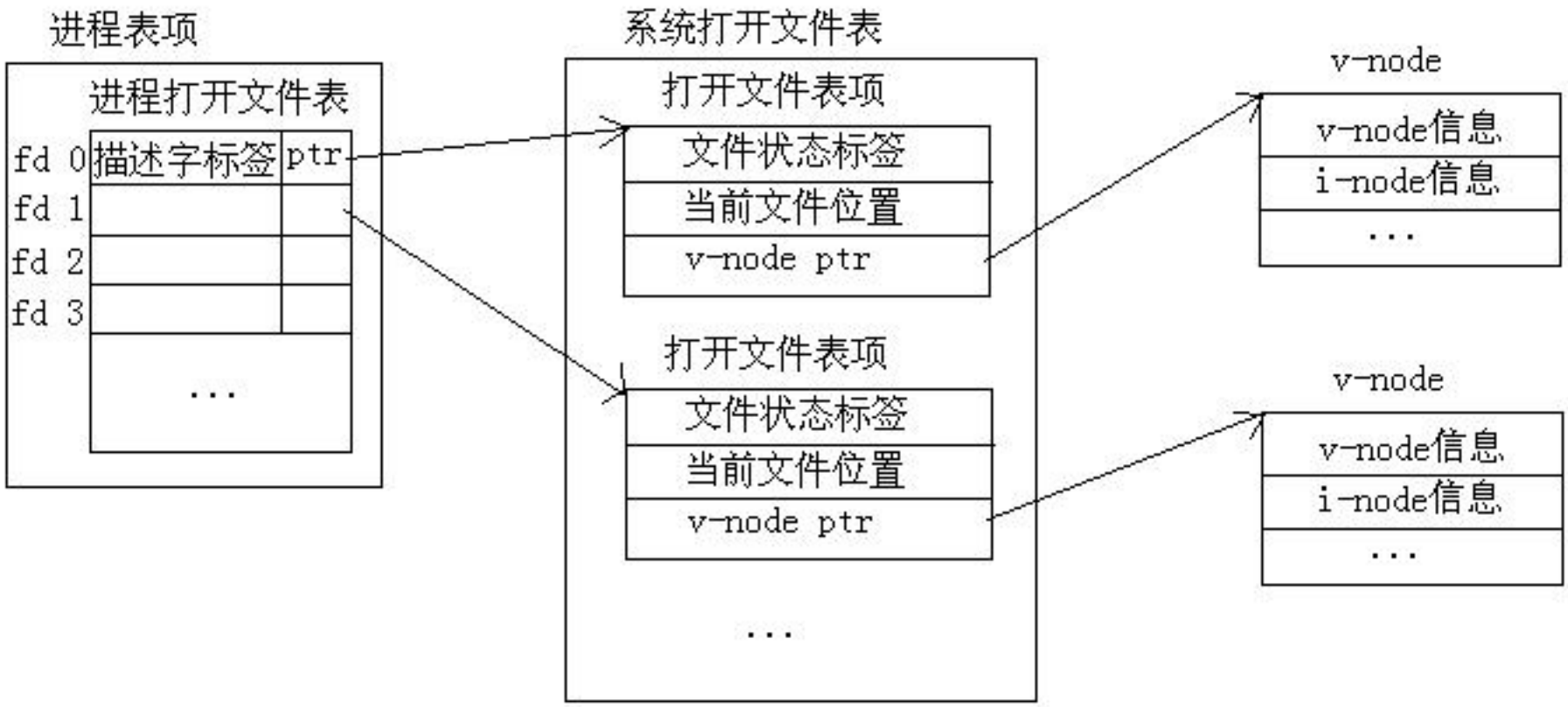
- 通过pipe系统调用创建无名管道，得到两个文件描述符
  - 文件描述符fildes[0]只能从管道读，fildes[1]只能向管道写；
  - 通过系统调用write和read进行管道的写和读；

```
#include <unistd.h>

int pipe(int filedes[2]);
```

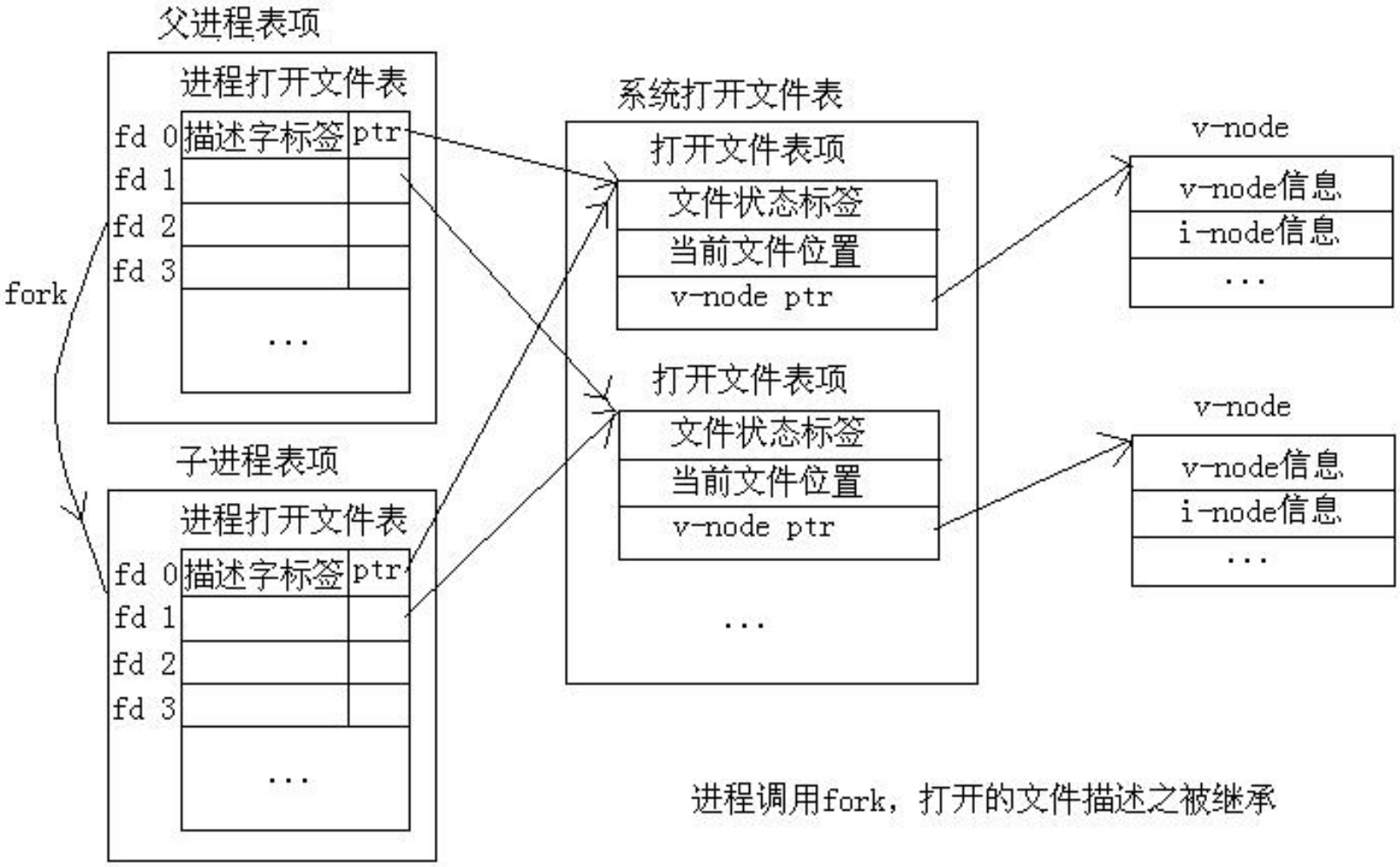
- 使用管道的两种限制：
  - 管道只能在具有亲缘关系的进程之间进行通信（具有公共祖先的进程之间）
    - 通过fork传递管道的描述符
  - 管道通信方式是单向的，进程间双向通信通常需要两个管道；

单个进程打开两个文件



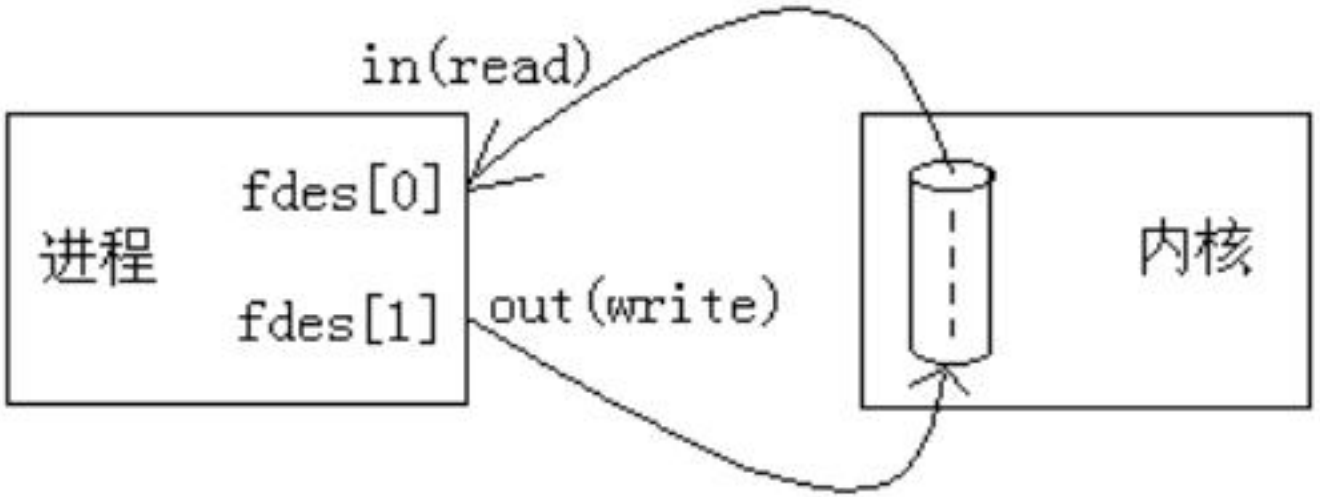
一个进程打开两个文件的内核数据结构

fork之后


父进程表项
进程打开文件表
fd 0
描述字标签
ptr
fd 1
fd 2
fd 3
...
fork
子进程表项
进程打开文件表
fd 0
描述字标签
ptr
fd 1
fd 2
fd 3
...
系统打开文件表
打开文件表项
文件状态标签
当前文件位置
v-node ptr
打开文件表项
文件状态标签
当前文件位置
v-node ptr
...
v-node
v-node信息
i-node信息
...
v-node
v-node信息
i-node信息
...
进程调用fork，打开的文件描述之被继承

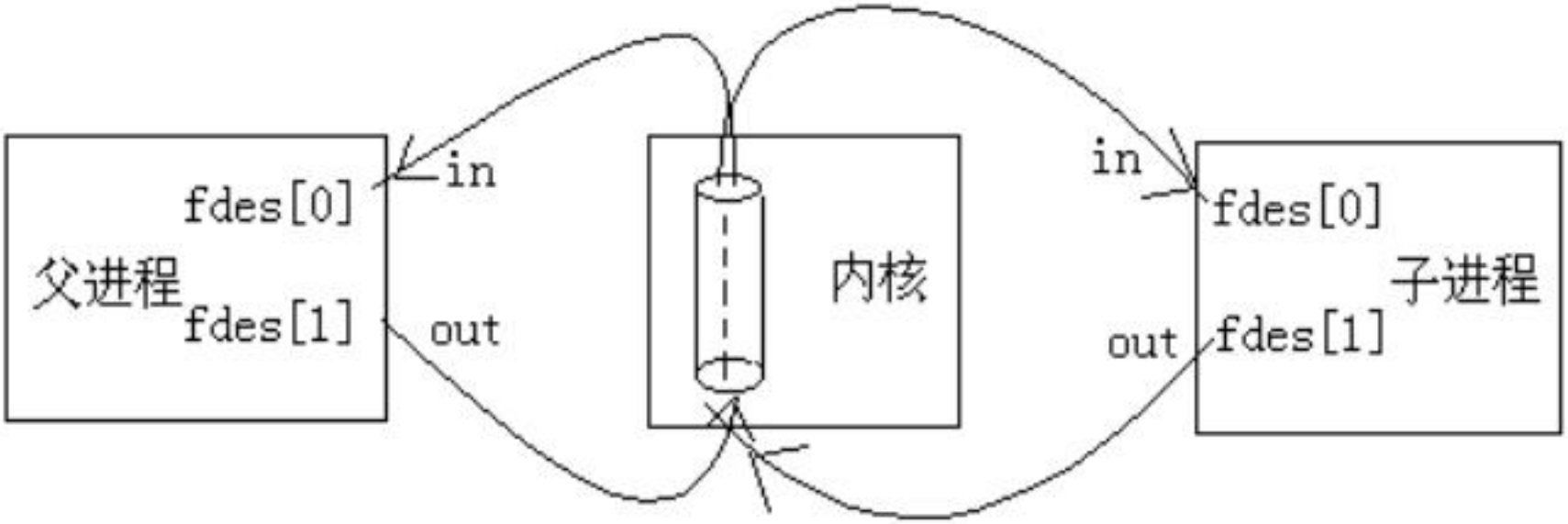
管道通信

- pipe函数成功后，内核打开两个文件描述字fdes[0], fdes[1]。fdes[0]输入端，fdes[1]为输出端。
- 当进程调用了pipe



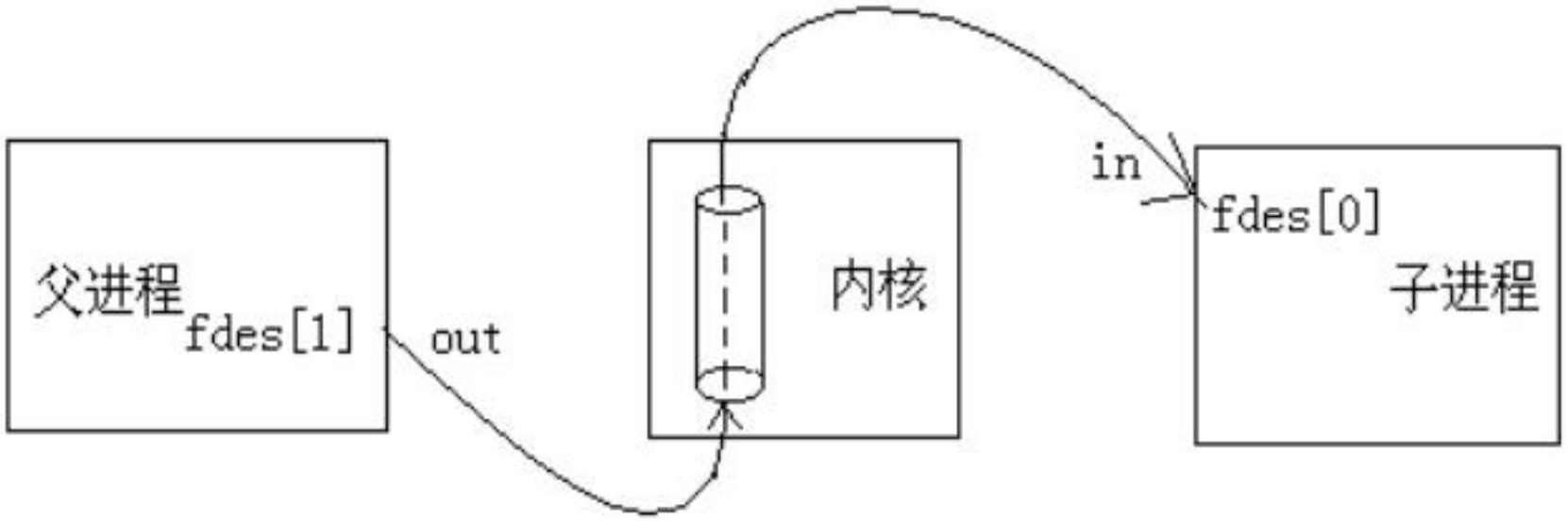
pipe创建的管道示意图

fork被调用后



fork之后共享一个管道的两个进程

两个进程分别关闭一个端



从父进程通往子进程的管道

```
int main(void) {
    pid_t pid;
    int fdes[2];
    if (pipe(fdes) < 0) { perror("pipe"); exit(1); }
    if ((pid = fork() < 0) { perror("fork"); exit(1); }
    if (pid > 0) {
        close(fdes[0]);
        write(fdes[1], "Hmmmmmmmmmmm",  12);
                        /*  1 2 3  4 5 6 7 8  9 0 1 2 */
    }
    else {
        char buf[4096]; ssize_t n;
        close(fdes[1])
        n = read(fdes[0], buf, 4096);
        if (n >= 0) { buf[n] = '\0'; printf("%s\n", buf); }
    }

    return 0;
```

管道破裂

- 如果一个管道的读端已经关闭，进程还继续向写端写数据，如：

```
pipe(fdes);
close(fdes[0]);
write(fdes[1], “Let me die”, 10);
```

则进程会收到一个SIGPIPE信号，表示管道破裂。默认动作为结束进程。

- 读一个写端已经关闭的管道则read返回0。

UNIX有名管道

- FIFO，有名管道
  - 特殊的文件类型：

    1，先入先出

    2，类似管道，在文件系统中不存在数据块，而是与一块内核缓冲区相关联

    3，有名字，FIFO的名字包含在系统的目录树结构中，可以按名访问
  - open，close，read，write等普通文件操作

```
#include <sys/types.h>
#include <sys/stat.h>

int mkfifo(const char *pathname, mode_t mode);
```

Example

进程A1、A2，。。。An1通过m个缓冲区向进程B1、B2、。。。Bn2不断发送消息。发送和接收工作遵循下列规则：

(1) 每个发送进程一次发送一个消息，写入一个缓冲区，缓冲区大小等于消息长度

(2) 对每个消息，B1，B2，Bn2都须各接收一次，读入各自的数据区内

(3) m个缓冲区都满时，发送进程等待，没有可读消息时，接收进程等待。

试用P、V操作组织正确的发送和接收工作。

Hint……

- 每个缓冲区只要写一次但要读n2次，因此，可以看成n2组缓冲区，每个发送者要同时写n2个缓冲区，而每个接收者只要读它自己的缓冲区：
- Sin[n2]=m;
- Sout[n2]=0;

解： 先将问题简化

- 设缓冲区的大小为1
- 有一个发送进程A1，有二个接收进程B1、B2
- 设有信号量Sin[1]、Sin[2] 初值为1
- 设有信号量Sout[1]、Sout[2] 初值为0

```
A1:
   while (1)
   {
       P(Sin[1]);
       P(Sin[2]);
         将数据放入缓冲区
       V(Sout[1]);
       V(Sout[2]);
   }
```

```
Bi:
   while (1)
   {
      P(Sout[i]);
          从缓冲区取数
      V(Sin[i]);
}
```

向目标前进一步

- 设缓冲区的大小为m
- 有一个发送进程A1
- 有二个接收进程B1、B2

- 设有信号量Sin[1]、Sin[2] 初值为m
- 设有信号量Sout[1]、Sout[2] 初值为0

```
A1:
   while (1)
   {
        P(Sin[1]);
        P(Sin[2]);
        P(mutex);
          将数据放入缓冲区
        V(mutex);
        V(Sout[1]);
        V(Sout[2]);
   }
```

```
Bi:
   while (1)
   {
      P(Sout[i]);
       P(mutex);
          从缓冲区取数
       V(mutex);
       V(Sin[i]);
};
```

到达目标

- 设缓冲区的大小为m
- 有n1个发送进程A1….An1
- 有n2个接收进程B1…Bn2

- 设有n2个信号量Sin[n2] 初值均为m
- 设有n2个信号量Sout[n2] 初值均为0

```
Aj:
while (1)
{
    for(i=1;i<=n2;i++)
     P(Sin[i]);
     P(mutex);
        将数据放入缓冲区
    V(mutex);
     for(i=1;i<=n2;i++)
     V(Sout[2]);
}
```

```
Bi :
  while (1)
  {
     P(Sout[i]);
      P(mutex);
         从缓冲区取数
      V(mutex);
      V(Sin[i]);
  };
```

利用直接通信原语解决生产者-消费者问题

```
repeat
        …
   produce an item in nextp;
        …
   send(consumer, nextp);
  until false;

  repeat
   receive(producer, nextc);
        …
   consume the item in nextc;
 until false;
```

What you need to do?

- 复习课本3. 6节的内容
- 课后作业： 习题16、33

See you next time!