

## 习题：

有一座东西方向的独木桥；用P,V操作分别实现以下要求：

- (1) 每次只允许一个人过桥；
- (2) 当独木桥上有自东向西的行人时，同方向的行人可以同时过桥，从西向东的方向，只允许一个人单独过桥；
- (3) 当独木桥上有行人时，同方向的行人可以同时过桥，相反方向的人必须等待。

互斥问题

(1) 每次只允许一个人过桥

```
semaphore mutex;
```

```
process P( )
```

```
{
```

```
    P(mutex);
```

```
    过桥;
```

```
    V(mutex);
```

```
}
```

(2) 当独木桥上有自东向西的行人时，同方向的行人可以同时过桥，从西向东的方向，只允许一个人单独过桥

相当于读者-写者问题

```
semaphore mutex = 1, DXmutex = 1;  
/*mutex用于东西方互斥，DXmutex用于东向西方向  
计数互斥*/
```

```
int DXcount = 0;  
/* DXcount用于东向西方向计数*/
```

## 第二章 进程的描述与控制

```
process P东向西( )  
{  
    P(DXmutex);  
    if(DXcount == 0 )  
        P(mutex);  
    DXcount = DXcount + 1;  
    V(DXmutex);  
    过桥;  
    P(DXmutex);  
    DXcount = DXcount - 1;  
    if(DXcount == 0 )  
        V(mutex);  
    V(DXmutex);  
}
```

```
process P西向东( )  
{  
    P(mutex);  
    过桥;  
    V(mutex);  
}
```

(3) 当独木桥上有行人时，同方向的行人可以同时过桥，相反方向的人必须等待

相当于读者-读者问题

```
semaphore mutex = 1, DXmutex = 1, XDmutex = 1;  
/*mutex用于东西方互斥，DXmutex用于东向西方向  
计数互斥，XDmutex用于西向东方向计数互斥*/
```

```
int DXcount = 0, XDcount = 0;  
/* DXcount用于东向西方向计数，XDcount用于西  
向东方向计数*/
```

```
process P东向西( )  
{  
    P(DXmutex);  
    if(DXcount == 0 )  
        P(mutex);  
    DXcount = DXcount + 1;  
    V(DXmutex);  
    过桥;  
    P(DXmutex);  
    DXcount = DXcount - 1;  
    if(DXcount == 0 )  
        V(mutex);  
    V(DXmutex);  
}
```

```
process P西向东( )  
{  
    P(XDmutex);  
    if(XDcount == 0 )  
        P(mutex);  
    XDcount = XDcount + 1;  
    V(XDmutex);  
    过桥;  
    P(XDmutex);  
    XDcount = XDcount - 1;  
    if(XDcount == 0 )  
        V(mutex);  
    V(XDmutex);  
}
```

## 2.6 进 程 通 信

- 进程通信——进程之间的信息交换。
- 进程之间的互斥和同步，交换的信息量少——**低级通信**。
- 信号量机制作为通信工具不够理想，表现在：
  - 效率低
  - 通信对用户不透明
- 本节介绍进程**高级通信**——是指用户可直接利用OS所提供的一组通信命令，高效地传送大量数据的一种通信方式。
- 高级通信过程对用户是**透明**的。大大减少了通信程序编制的复杂性。

进程之间通信所需要的共享数据结构的设计、数据的传送、进程的互斥与同步等等，都需要程序员去实现



高级通信工具最主要的特点是：

(1) 使用方便。OS隐藏了实现进程通信的具体细节，向用户提供了一组用于实现高级通信的命令（原语），用户可方便地直接利用它实现进程之间的通信——通信过程对用户是透明的。这样就大大减少了通信程序编制上的复杂性。

(2) 高效地传送大量数据。用户可直接利用高级通信命令（原语）高效地传送大量的数据。



### 2.6.1 进程通信的类型

#### 1. 共享存储器系统 (Shared-Memory System)

在共享存储器系统中，相互通信的进程共享某些数据结构或共享存储区，进程之间能够通过这些空间进行通信。据此，又可把它们分成以下两种类型：

- (1) 基于共享数据结构的通信方式。
- (2) 基于共享存储区的通信方式。

### (1) 基于共享数据结构的通信方式

如生产者-消费者问题中，是用有界缓冲区这种数据结构来实现通信的。

公用数据结构的设置及对进程间同步的处理，都是程序员的职责，增加了程序员的负担，而操作系统只需提供共享存储器。——低效的通信方式（低级通信）

(2) 基于共享存储区的通信方式：在存储区中划出一块共享存储区，诸进程可通过对共享存储区中数据的读和写来实现通信。——属于高级通信方式。

- 先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；
- 若系统已经将该分区分配给其它进程，则将其描述符返回给申请者；
- 申请者将获得的共享存储分区连接到本进程上；
- 此后，便可像读写普通存储器那样地读写该公用存储分区。——UNIX/LINUX与之有关的系统调用有4个

(2) 基于共享存储区的通信方式：在存储区中划出一块共享存储区，诸进程可通过对共享存储区中数据的读和写来实现通信。——属于高级通信方式。

- 先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；

- 若系统已经将该分区分配给其他进程，则按前所述

`shmget()`, `shmat()`, `shmdt()`, `shmctl()`

- 此后，便可像读写普通存储器那样地读写该公用存储分区。——UNIX/LINUX与之有关的系统调用有4个

## ■ 2. 管道（pipe）通信

管道——用于连接一个读进程和一个写进程以实现它们之间通信的一个共享文件，又称pipe文件。

```
rjxy@localhost:/mnt/hgfs/share/lesson 3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
[root@localhost lesson 3]# ls /etc
abrt                hosts.allow          prelink.cache
acpi                hosts.deny            prelink.conf
adjtime             hp                   prelink.conf.d
aliases             httpd                 printcap
aliases.db          idmapd.conf          profile
alsa               init                 profile.d
alternatives        init.d               protocols
anacrontab          inittab              pulse
anthy-conf          inputrc              quotagrpadmins
asound.conf         ipa                  quotatab
at.deny             iproute2             rc
audisp              issue                rc0.d
audit               issue.net             rc1.d
autofs_ldap_auth.conf java                  rc2.d
auto.master         jvm                  rc3.d
auto.misc           jvm-common           rc4.d
auto.net            kde                   rc5.d
auto.smb            kdump-adv-conf       rc6.d
avahi               kdump.conf           rc.d
bash_completion.d  krb5.conf            rc.local
bashrc              latrace.conf         rc.sysinit
blkid               latrace.d            readahead.conf
bluetooth           ld.so.cache          redhat-lsb
```

```
rjxy@localhost:/mnt/hgfs/share/lesson 3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
[root@localhost lesson 3]# ls /etc |grep ab
abrt
anacrontab
crontab
crypttab
fstab
inittab
mtab
quotatab
rwtab
rwtab.d
statetab
statetab.d
[root@localhost lesson 3]#
```

## ■ 2. 管道（pipe）通信

管道——用于连接一个读进程和一个写进程以实现它们之间通信的一个共享文件，又称pipe文件。

- ◆ 写进程以字节流的形式将大量数据送入管道；
- ◆ 读进程从管道中接收（读）数据。

为协调双方的通信，管道机制必须提供以下三方面的协调能力：

- ★ 互斥—— 当一个进程正在对pipe执行读/写操作时，其它进程必须等待；
- ★ 同步—— 当写进程把一定数据写入pipe，便去睡眠等待，直至读进程取走数据后，再把它唤醒；当读进程读一空pipe时，也应睡眠等待，直至写进程将数据写入pipe后，再把它唤醒。
- ★ 确定对方是否存在—— 只有确定对方已存在时，才能进行通信。



### ■ 3. 消息传递系统（Message passing system）

消息传递机制——用得最广泛的一种高级通信机制，包括单机系统、多机系统、计算机网络等。

- 进程间的数据交换，是以格式化的消息(message)为单位的。
- 计算机网络中，message又称为报文。
- 利用系统提供的一组通信命令（原语）进行通信。
- OS隐蔽了通信实现细节，大大简化了通信程序编制的复杂性，因而得到广泛应用。

■ 直接通信方式

■ 间接通信方式



### ■ 3. 消息传递系统（Message passing system）

消息传递机制——用得最广泛的一种高级通信机制，包括单机系统、多机系统、计算机网络等。

● 进程间的数据交换，是以格式化的消息(message)为单位的。

- 直接通信方式：发送进程利用OS所提供的发送原语，直接把消息发送给目标进程；
- 间接通信方式：发送和接收进程，都通过共享中间实体（称为邮箱）的方式进行消息的发送和接收，完成进程间的通信

■ 直接通信方式

■ 间接通信方式

## ■ 4. 客户机-服务器系统

最早用于单机系统，解决多进程同时通信时端口和物理线路的多路复用问题，目前已成为主流的网络通信实现机制。

主要包括：

- 套接字
- 远程过程调用和远程方法调用

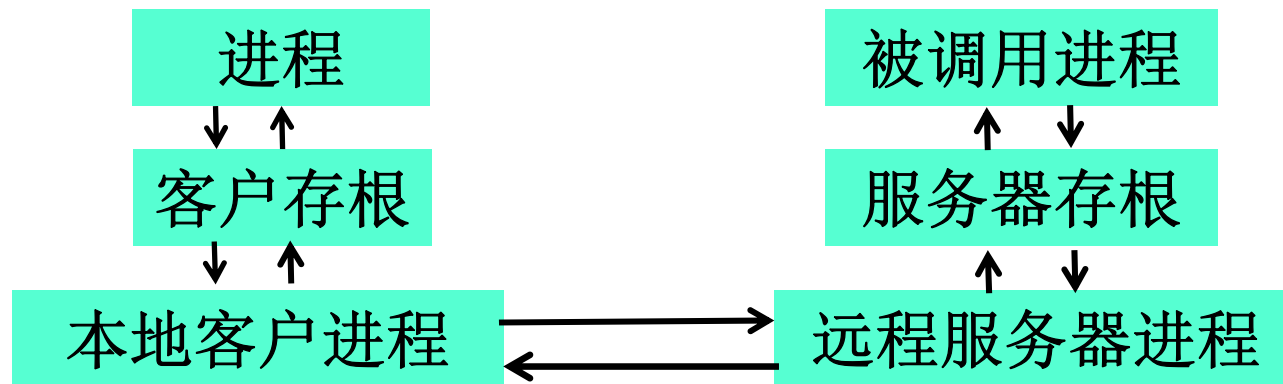
### 1) 套接字（**socket**）

一个套接字就是一个通信标示类型的数据结构。套接字分两类：

- 基于文件型：同一台主机环境，一个套接字关联一个特殊文件，类似管道。
- 基于网络型：不同主机的网络环境。被分配一对套接字，一个属于接收进程（或服务器），一个属于发送进程（或客户端）。

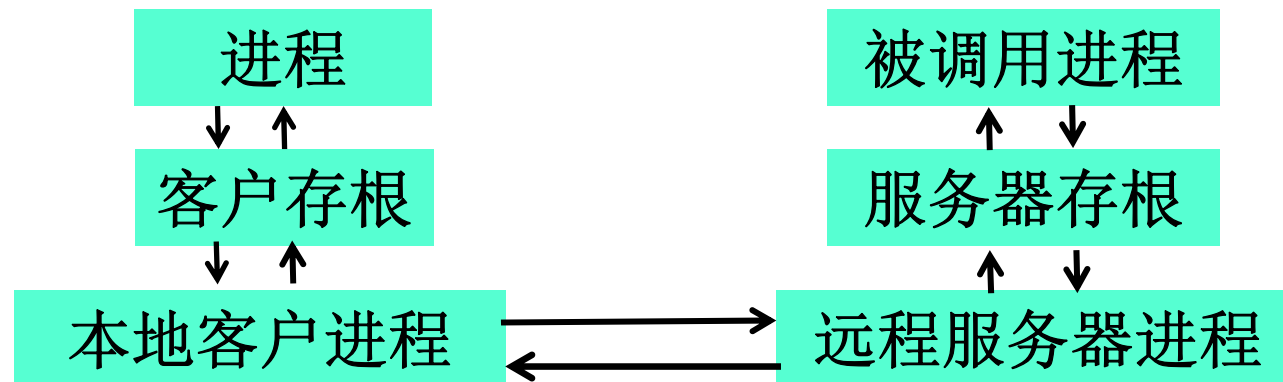
## 2) 远程过程调用 (RPC, Remote Procedure Call) :

是一个通信协议，用于通过网络连接的系统。该协议允许一台主机系统上的进程调用另外一台主机系统上的进程，程序员表现为常规的过程调用，无需额外地为此编程。如果涉及的软件采用面向对象编程，远程过程调用亦可以成为远程方法调用。

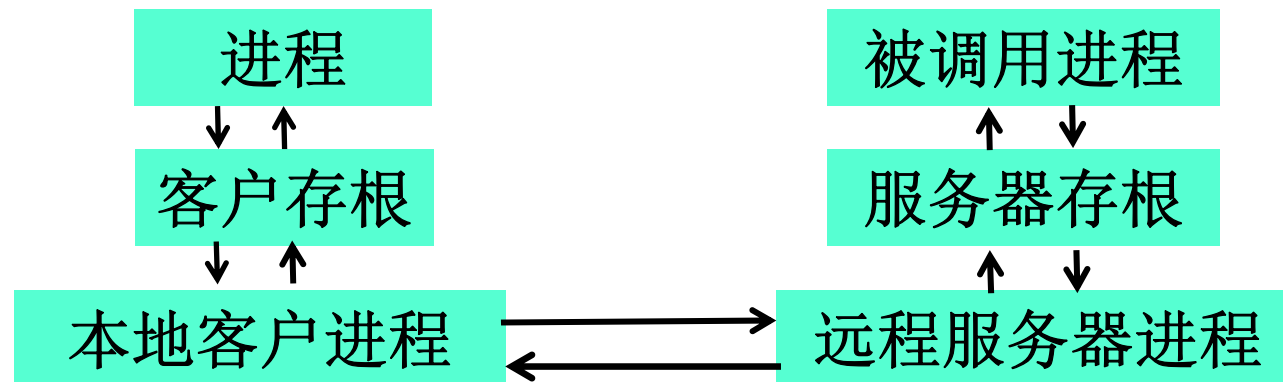


## 第二章 进程的描述与控制

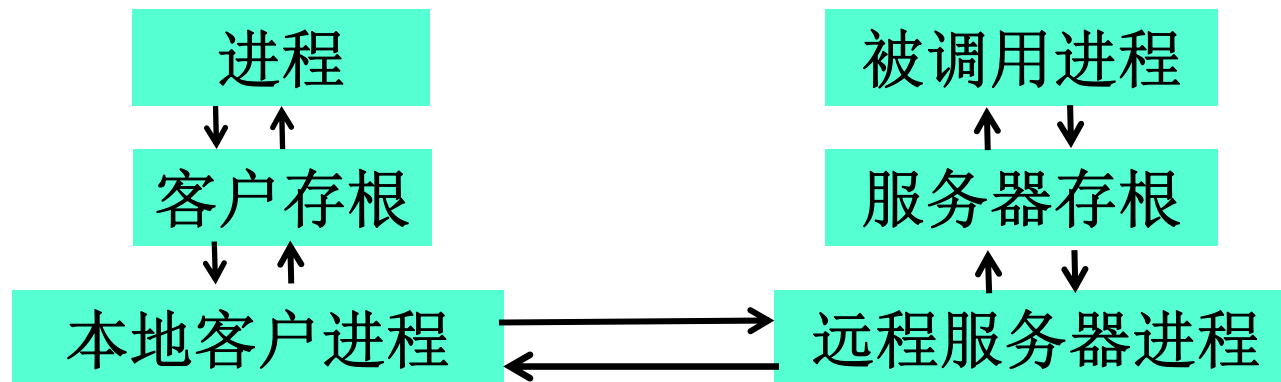
- (1) 本地过程调用者以一般方式调用远程过程在本地关联的客户存根，传递相应的参数，然后将控制权转移给客户存根；
- (2) 客户存根执行，完成包括过程名和调用参数等信息的消息建立，将控制权转移给本地客户进程；
- (3) 本地客户进程完成与服务器的消息传递，将消息发送到远程服务器进程；



- (4) 远程服务器进程接收消息后转入执行，并根据其中的远程过程名找到对应的服务器存根，将消息转给该存根；
- (5) 该服务器存根接到消息后，由阻塞状态转入执行状态，拆开消息从中取出过程调用的参数，然后以一般方式调用服务器上关联的过程；

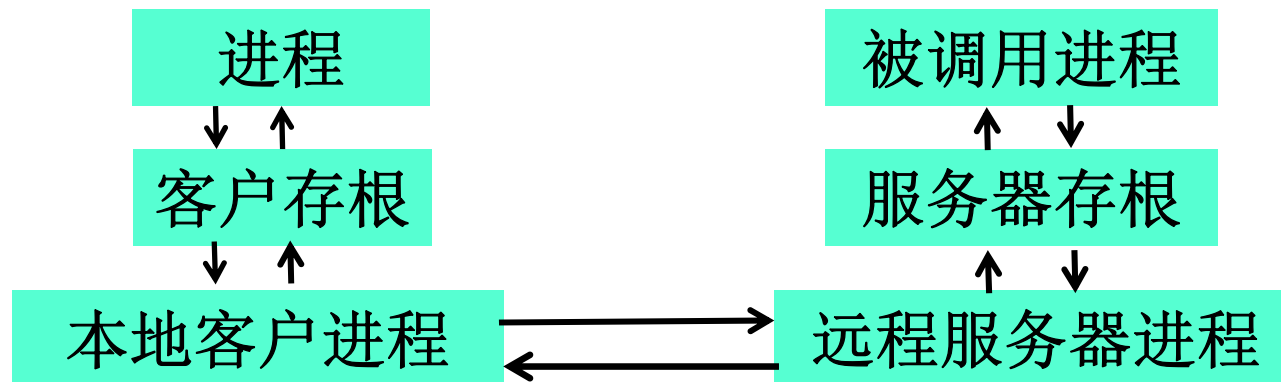


- (6) 在服务器端的远程过程运行完毕后，将结果返回给与之关联的服务器存根；
- (7) 该服务器存根获得控制权运行，将结果打包为消息，并将控制权转移给远程服务器进程；
- (8) 远程服务器进程将消息发送回客户端；





- (9) 本地客户进程接收到消息后，根据其中的过程名将消息存入关联的客户存根，再将控制权转移给客户存根；
- (10) 客户存根从消息中取出结果，返回给本地调用者进程，并完成控制权的转移。





## ■ 2.6.2 消息传递通信的实现方法

### 1. 直接通信方式

#### 1) 直接通信原语

是指发送进程利用OS所提供的命令，直接

要求发送和接收进程都以显式方式

两条通信原语为：

send (Receiver, message) ;

receive (Sender, message) ;

发送一个消息message给接收进程Receiver

接收Sender发来的消息message

#### 对称寻址方式

不足：一旦改变进程的名称，则可能需要检查所有其他进程的定义，有关该进程旧名称的所有引用都必须查找到，以便将其修改为新名称。不利于实现进程定义模块化。

## ■ 2.6.2 消息传递通信的实现方法

### 1. 直接通信方式

#### 1) 直接通信原语

是指发送进程利用OS所提供的命令，直接把消息发送给目标进程。

要求发送和接收进程都以显式方式提供对方的标识符。

两条通信原语为：

send (Receiver, message) ;

receive (Sender, message) ;

接收Sender发来的  
消息message

有时，接收进程可能与多个发送进程通信，故不可能事先知道发送进程。对于这样的应用，接收原语中的源进程参数，是完成通信后的返回值。接收原语可表示为：

receive (id, message) ; id是返回值（标识符）

## ■ 2.6.2 消息传递通信的实现方法

### 1. 直接通信方式

#### 1) 直接通信原语

是指发送进程利用OS所提供的命令，直接把消息发送给目标进程。

要求发送和接收进程都以显式方式提供对方的标识符。

两条通信原语为：

#### 非对称寻址方式

有时，接收进程可能与多个发送进程通信，故不可能事先知道发送进程。对于这样的应用，接收原语中的源进程参数，是完成通信后的返回值。接收原语可表示为：

`receive (id, message) ;` `id`是返回值（标识符）

### 2) 消息的格式

在单机系统环境中，由于发送进程和接收进程处于同一台机器中，有着相同的环境，所以消息的格式比较简单，可采用比较短的定长消息格式，以减少对消息的处理和存储开销。该方式可用于办公自动化系统中，为用户提供快速的便笺式通信。但这种方式对于需要发送较长消息的用户是不方便的。

为此，可采用变长的消息格式，即进程所发送消息的长度是可变的。对于变长消息，系统无论在处理方面还是存储方面，都可能会付出更多的开销，但其优点在于方便用户。

### 3) 进程的同步方式

- 发送进程和接收进程均阻塞

用于进程之间紧密同步，发送进程和接收进程之间无缓冲时。

- 发送进程不阻塞、接收进程阻塞

应用最广的一种进程同步方式。平时发送进程不阻塞，可以尽快把一个或多个消息发送给多个目标；接收进程阻塞，直到发送进程发来消息时才被唤醒。

- 发送进程和接收进程均不阻塞

较常见的一种进程同步形式。平时发送进程和接收进程都忙自己的事情，仅当发生某事件使它们无法继续运行时，才把自己阻塞起来等待。

### 4) 通信链路

为使在发送进程和接收进程之间能进行通信，必须在两者之间建立一条通信链路。有两种方式建立通信链路。

第一种方式是：由发送进程在通信之前用显式的“建立连接”命令（原语）请求系统为之建立一条通信链路，在链路使用完后拆除链路。一般用于计算机网络中。

第二种方式是：发送进程无需明确提出建立链路的请求，只需利用系统提供的发送命令（原语），系统会自动地位置建立一条链路。一般用于单机系统中。



## 2. 信箱（邮箱）通信

进程之间的通信，需要通过作为共享数据结构的实体来完成，这种中间实体称为信箱。

- ▲ 信箱暂存发送进程发送给目标进程的消息；
- ▲ 接收进程从信箱中取出对方发给自己的消息；
- ▲ 每个信箱都有一个唯一的标识符；
- ▲ 消息在信箱中可以安全地保存，只允许核准的目标用户随时读取。

既可实现实时通信，也可实现非实时通信。



### 1) 信箱的结构

信箱定义为一种数据结构。在逻辑上，可以将其分为两个部分：

(1) 信箱头——存放有关信箱的描述信息，如信箱标识符、信箱的拥有者、信箱口令等。

(2) 信箱体——由若干个可以存放消息的信箱格组成，信箱格的数目以及每格的大小是在创建信箱时确定的。

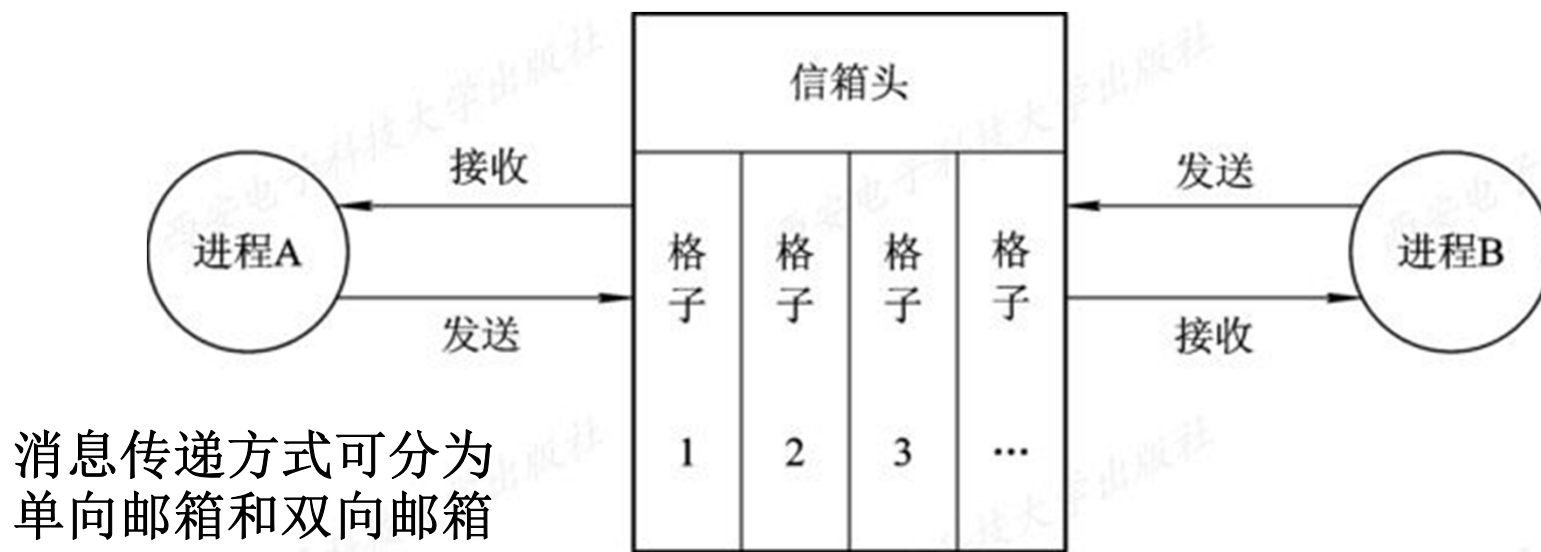


图2-16 双向信箱示意图

## 2) 信箱通信原语

系统为邮箱通信提供了若干条原语，分别用于：

- (1) 邮箱的创建和撤消。
- (2) 消息的发送和接收。

`Send(mailbox, message);` 将一个消息发送到指定邮箱

`Receive(mailbox, message);` 从指定邮箱接收一个消息

### 3) 信箱的类型

邮箱可由操作系统创建，也可由用户进程创建，创建者是邮箱的拥有者。据此，可把邮箱分为以下三类：

- (1) 私用邮箱
- (2) 公用邮箱
- (3) 共享邮箱

用户进程可为自己建立一个新邮箱，并作为该进程的一部分。邮箱的拥有者有权从邮箱中读取消息，其他用户只能将自己构成的消息发送到该邮箱中。

私有邮箱可以采用单向通信链路的邮箱来实现

拥有该邮箱的进程结束时，邮箱也随之消失

### 3) 信箱的类型

邮箱可由操作系统创建，也可由用户进程创建，创建者是邮箱的拥有者。据此，可把邮箱分为以下三类：

- (1) 私用邮箱
- (2) 公用邮箱
- (3) 共享邮箱

由操作系统创建，并提供给系统中的所有核准进程使用。

核准进程既可以把消息发送到邮箱中，也可以从邮箱中读取发送给自己的消息。

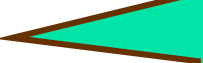
公用邮箱应采用双向通信链路的邮箱来实现

公用邮箱在系统运行期间始终存在

### 3) 信箱的类型

邮箱可由操作系统创建，也可由用户进程创建，创建者是邮箱的拥有者。据此，可把邮箱分为以下三类：

- (1) 私用邮箱
- (2) 公用邮箱
- (3) 共享邮箱



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。

邮箱的拥有者和共享者都有权从邮箱中取走发给自己的消息

利用邮件通信时，发送进程和接收进程之间存在以下四种关系：

- (1) 一对一关系
- (2) 多对一关系
- (3) 一对多关系
- (4) 多对多关系

发送进程和接收进程建立一条两者专用的通信链路，使两者之间的交互不受其他进程的干扰。

利用邮件通信时，发送进程和接收进程之间存在以下四种关系：

- (1) 一对一关系
- (2) 多对一关系
- (3) 一对多关系
- (4) 多对多关系

允许提供服务的进程与多个用户进程之间进行交互，也称为客户/服务器交互



利用邮件通信时，发送进程和接收进程之间存在以下四种关系：

- (1) 一对一关系
- (2) 多对一关系
- (3) 一对多关系
- (4) 多对多关系

允许一个发送进程与多个接收进程交互，使发送进程可用广播方式向接收者（多个）发送消息。

利用邮件通信时，发送进程和接收进程之间存在以下四种关系：

- (1) 一对一关系
- (2) 多对一关系
- (3) 一对多关系
- (4) 多对多关系

允许建立一个公共邮箱，让多个进程都能像邮箱中投递消息，也可从邮箱中取走属于自己的消息。

### 2.6.3 直接消息传递系统实例

消息缓冲队列通信机制首先由美国的Hansan提出，并在RC 4000系统上实现，后来被广泛应用于本地进程之间的通信中。在这种通信机制中，发送进程利用Send原语将消息直接发送给接收进程；接收进程则利用Receive原语接收消息。

## 1. 消息缓冲队列通信机制中的数据结构

(1) 消息缓冲区 其结构可描述如下：

```
typedef struct message_buffer {  
    long sender;           //发送者进程标识符  
    int size;              //消息长度  
    char text[N];          //消息正文  
    struct message_buffer * next; //指向下一个消息缓冲  
                               区的指针  
};
```

### (2) PCB中有关通信的数据项

采用消息缓冲队列通信机制时，除了需要为进程设置消息缓冲队列外，还应在进程的**PCB**中增加消息队列队首指针，以及用于实现同步的互斥信号量**mutex**和资源信号量**sm**。

```
typedef struct processcontrol_block {  
    struct message_buff *mq;      //消息队列首指针  
    semaphore mutex;              //消息队列互斥信号量  
    semaphore sm;                 //消息队列资源信号量  
}PCB;
```

## 2. 发送原语

发送进程在利用发送原语发送消息之前，应先在自己的内存空间设置一发送区a，如图2-17所示，把待发送的消息正文、发送进程标识符、消息长度等信息填入其中，然后调用发送原语，把消息发送给目标（接收）进程。发送原语首先根据发送区a中所设置的消息长度a.size来申请一缓冲区i，接着，把发送区a中的信息复制到缓冲区i中。为了能将i挂在接收进程的消息队列mq上，应先获得接收进程的内部标识符j

（PID），然后通过insert操作将i挂在j.mq上。由于该队列属于临界资源，故在执行insert操作的前后都要执行P和V操作

。



## 第二章 进程的描述与控制

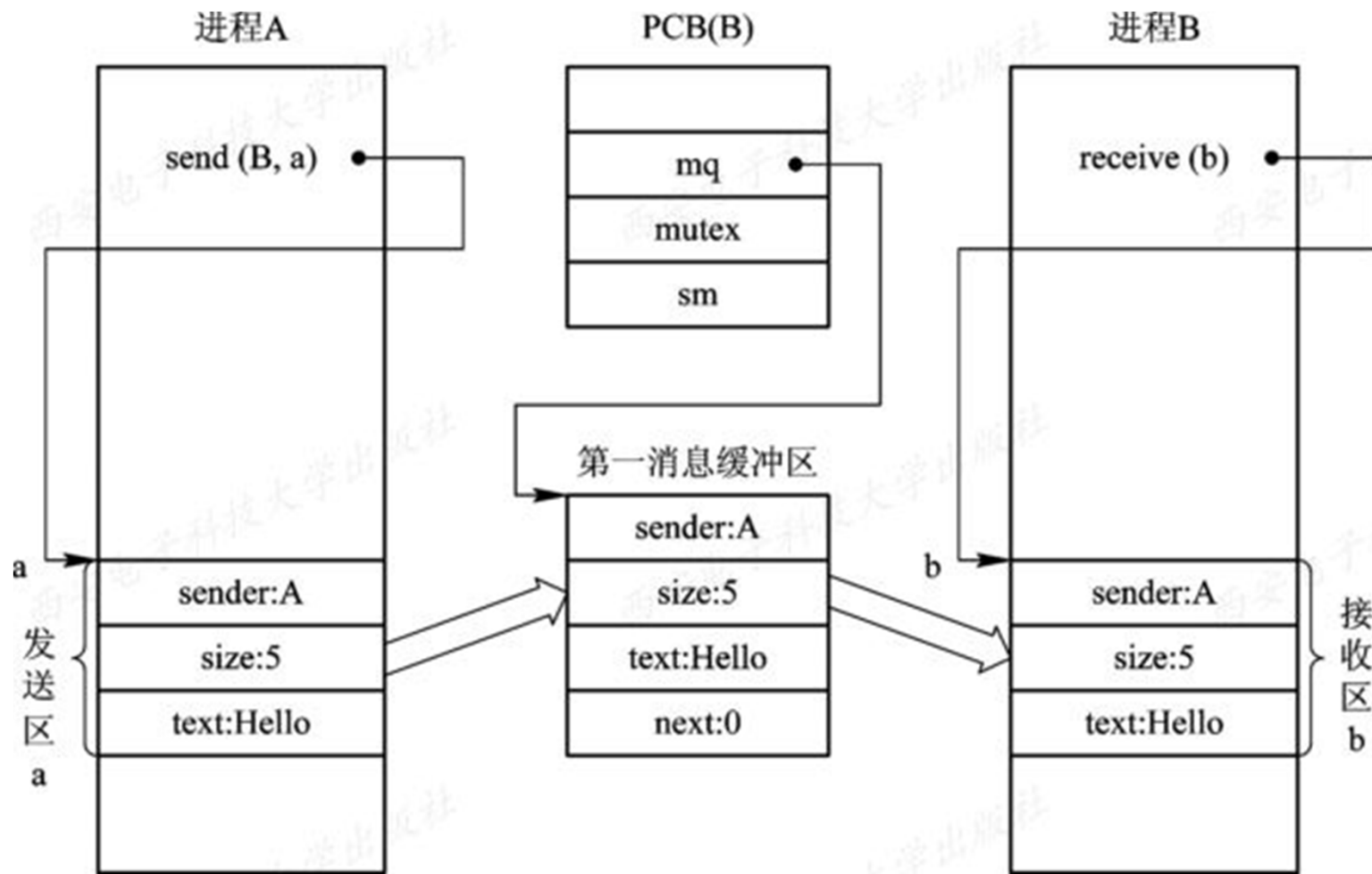


图2-17 消息缓冲通信

## 第二章 进程的描述与控制

```
send(receiver,a){ //receiver为接收  
首地址
```

```
    getbuf(a.size,i); //根据a.size申请缓
```

```
    i.sender = a.sender;
```

```
    i.size = a.size;
```

```
    i.text = a.text; //将发送区a中的
```

中

```
    i.next = 0;
```

```
    getid(PCBset,receiver.j); //获得接收进程内部标识符
```

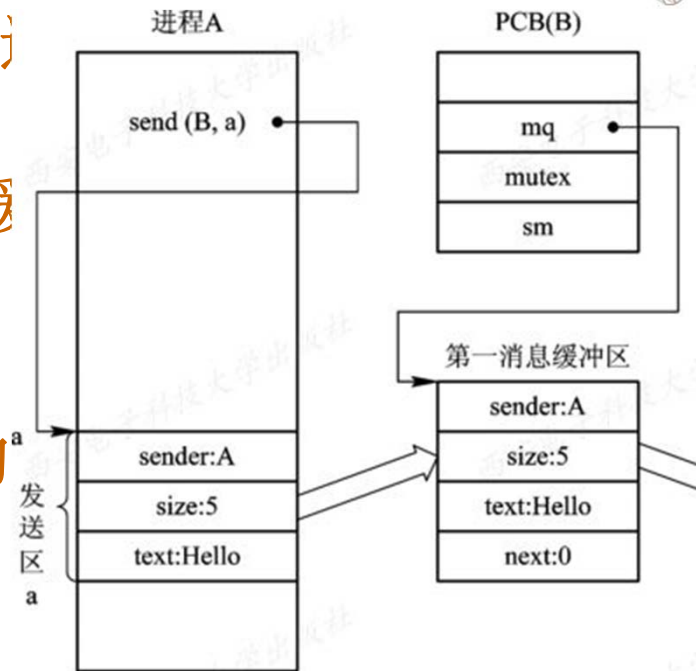
```
    P(j.mutex);
```

```
    insert(j.mq,i); //将消息缓冲区插入到消息队列中
```

```
    V(j.mutex);
```

```
    V(j.sm); //资源数目增1(有可能唤醒接收进程)
```

```
}
```



### 3. 接收原语: receive(b)

```
void receive(b) {
```

从自己的消息缓冲队列mq中，摘下第一个消息缓冲区i，并将其中的数据复制到以b为首址的指定消息接收区内。

```
    b.sender = i.sender
```

```
    b.size = i.size
```

```
    b.text = i.text
```

```
    releasebuf(i);
```

```
}
```

//j为

//将j

将消息缓冲区i中消息复制到接收区b

//释放消息缓冲区

