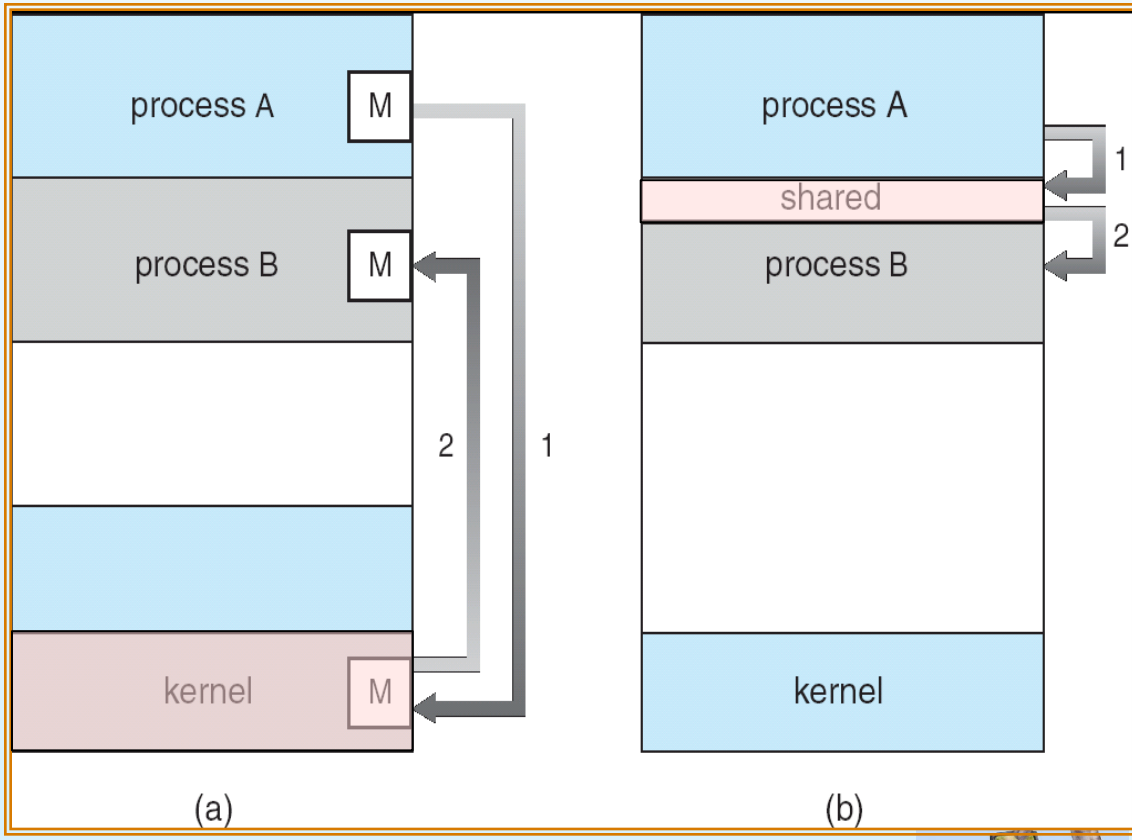
4、进程间通信

协同进程

- 独立进程：不会影响另一个进程的执行或被另一个进程执行影响
- 协同进程：可能影响另一个进程的执行或被另一个进程执行影响
- 进程协同的优点
  - 信息共享
  - 加速运算
  - 模块化
  - 方便

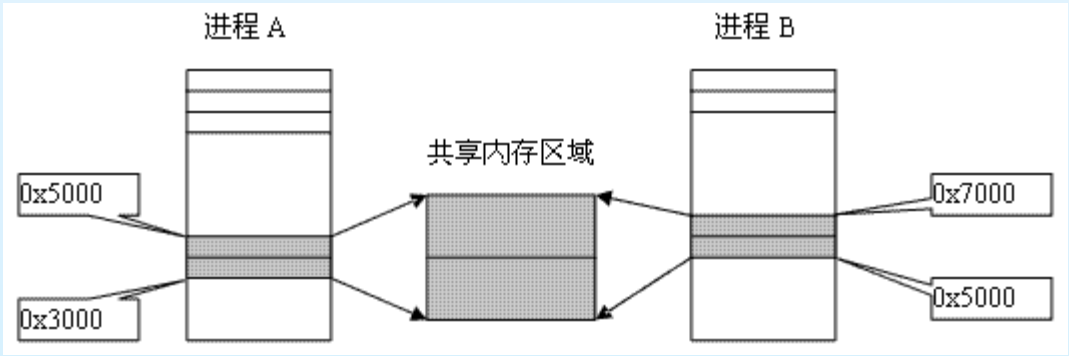
进程间通信(IPC)

- 用于进程通信的机制，同步其间的活动
- 两种基本模式：
  - 共享内存
    - 以最快的速度进行方便的通信
  - 消息传递
    - 交换较少数量的数据



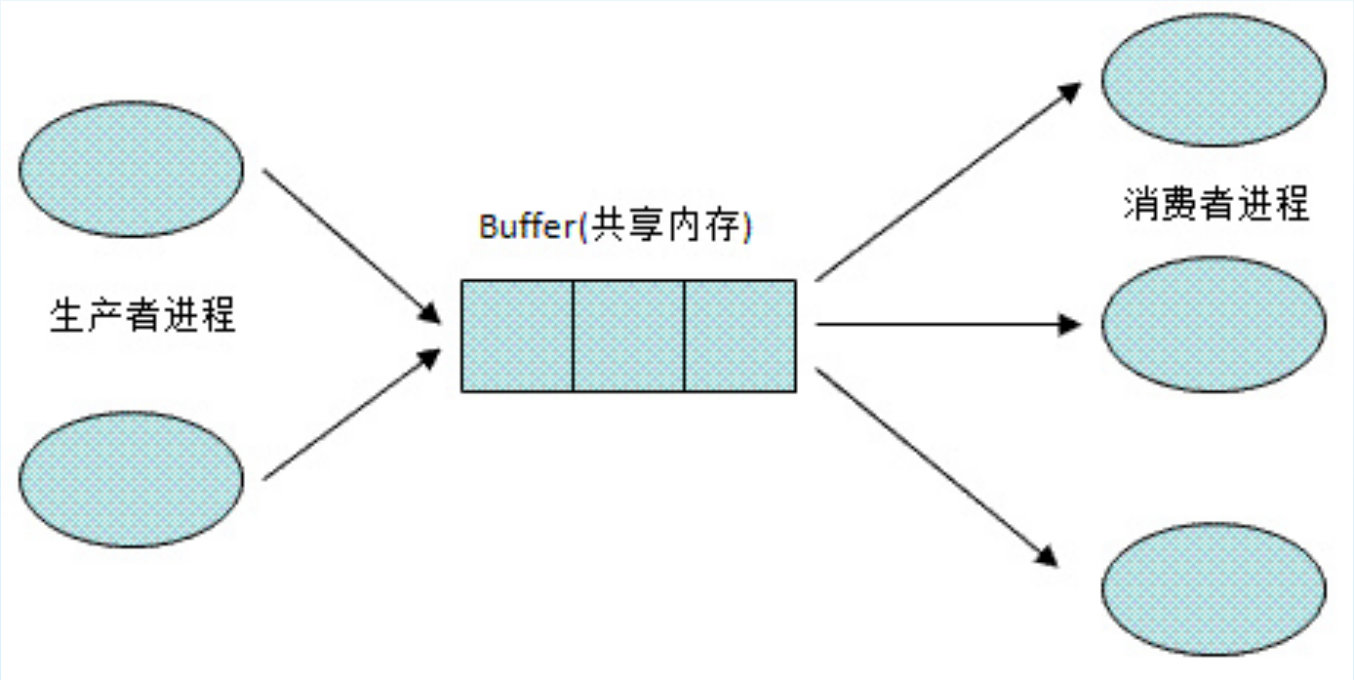
共享内存

- 一块内存在多个进程间共享
- 通信由应用程序自己控制
- 一般用于大数据通信
- 实现手段：
  - 文件映射
  - 管道
  - 剪贴板



例子：生产者–消费者

- 生产者进程生产，供消费者进程消费的信息
  - 无界缓冲(Unbounded-buffer)没有对缓冲区大小的限制
  - 有界缓冲(Bounded-buffer)对缓冲区大小作了限定



有界缓冲

- Shared data

```
#define BUFFER_SIZE 10
Typedef struct {
   . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- in指向缓冲区中下一个空位；out指向缓冲区中第一个非空位
- 但最多只能填满缓冲区的BUFFER_SIZE-1个项

生产者进程

```
item next_produced;
while (true) {
   /* Produce an item in next_produced*/
      while (((in = (in + 1) % BUFFER SIZE count)  == out)
               ; /* do nothing - no free buffers */
      buffer[in] = next_produced;
      in = (in + 1) % BUFFER SIZE;
}
```

消费者进程

```
item next_consumed;
while (true) {
    while (in == out)
        ; // do nothing -- nothing to consume

    // remove an item from the buffer
    next_consumed = buffer[out];
    out = (out + 1) % BUFFER SIZE;
    return item;
}
```

消息传递

- 消息传递在微内核中的应用
- 远程通信无法采用共享内存
- 两个原子操作 ：
  - 发送**send**(*message*) – 固定或可变大小消息
  - 接收**receive**(*message*)
- 若*P*与*Q*要通信，需要：
  - 建立通信连接
  - 通过send/receive交换消息
- 通信连接的实现
  - 物理的（如，共享存储，硬件总线）
  - 逻辑的（如，逻辑特性）

消息传递实现问题

- 连接如何建立？
- 连接可同多于两个的进程相关吗？
- 每对在通信进程有多少连接？
- 一个连接的容量是多少？
- 连接可使用的固定或可变消息的大小？
- 连接是单向的还是双向的？

直接通信

- 进程必须显式的命名接受者和发送者
  - **send** (*P, message*) –向进程P发消息
  - **receive**(*Q, message*) –从进程Q收消息
- 通信连接的特性
  - 连接自动建立
  - 连接精确地与一对通信进程相关
  - 在每一对通信进程间存在一个连接
  - 连接可单向，但通常双向

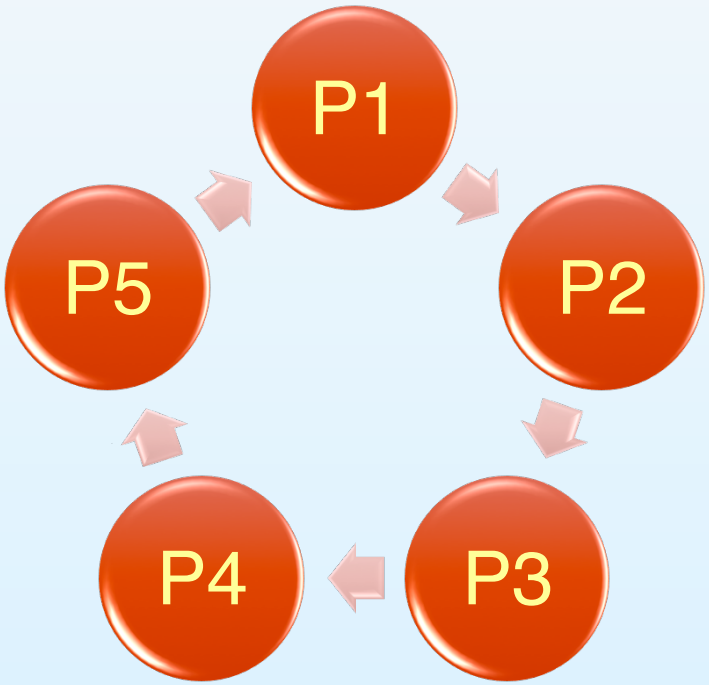
间接通信

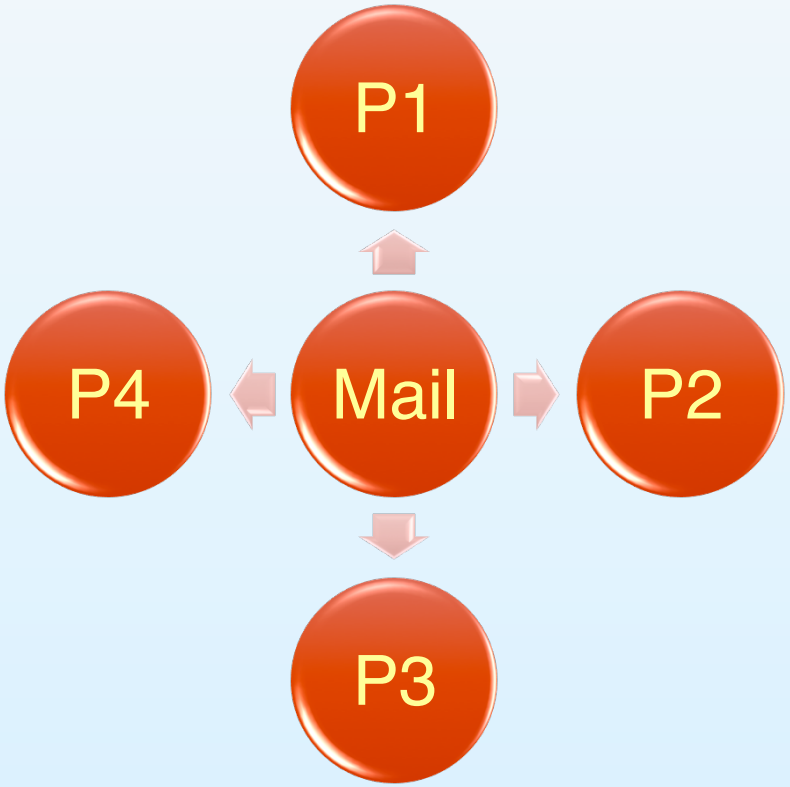
- 消息导向至信箱并从信箱接收
  - 每一个信箱有一个唯一的id
  - 仅当共享一个信箱时进程才能通信
- 通信连接的特性
  - 仅当进程共有一个信箱时连接才能建立
  - 连接可同多个进程相关
  - 每一对进程可共享多个通信连接
  - 连接可是单向或双向的

间接通信

- 操作
  - 创建新的信箱
  - 通过信箱发送和接收消息
  - 销毁信箱
- 两个原语被定义
  - **send**(*A, message*) – 发送消息到信箱 A
  - **receive**(*A, message*) – 从信箱 A接收消息

直接与间接的比较





间接通信

- 信箱共享
  - P_1, P_2与P_3共享信箱A
  - P_1发送；　P_2与P_3接受
  - 谁得到消息？
- 解决方案
  - 允许一个连接最多同两个进程相关
  - 只允许一个时刻有一个进程执行接受操作
  - 允许系统任意选择接收者。发送者被通知谁是接收者。

同 步

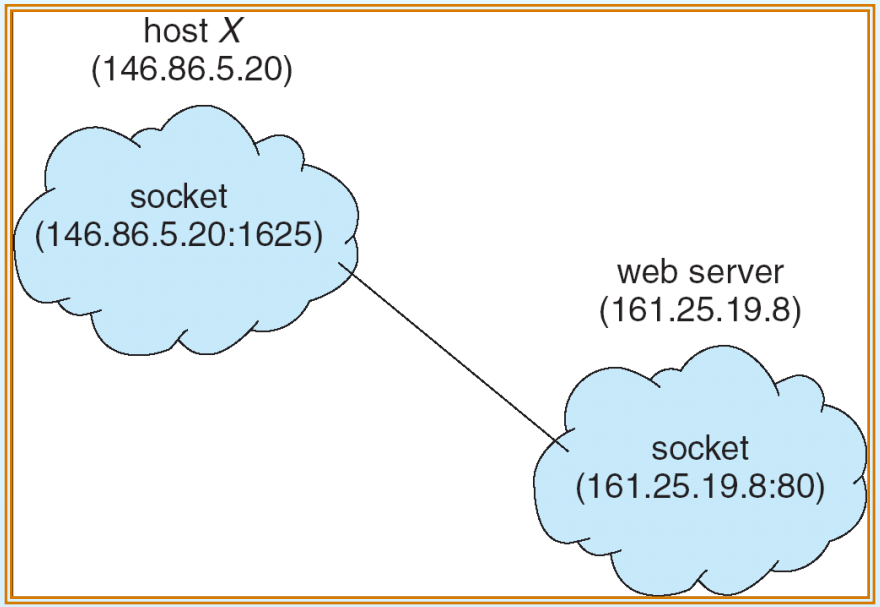
- 消息传递可阻塞（blocking）或非阻塞（non-blocking），也称为同步或异步
- 阻塞-同步
  - 阻塞send：发送进程阻塞，直到消息被接收
  - 阻塞receive：接受者进程阻塞，直到有消息可用
- 非阻塞-异步
  - 非阻塞send：发送进程发送消息并继续操作
  - 非阻塞receive: 接收者收到一个有效消息或无效消息

远程通信-客户机服务器通信

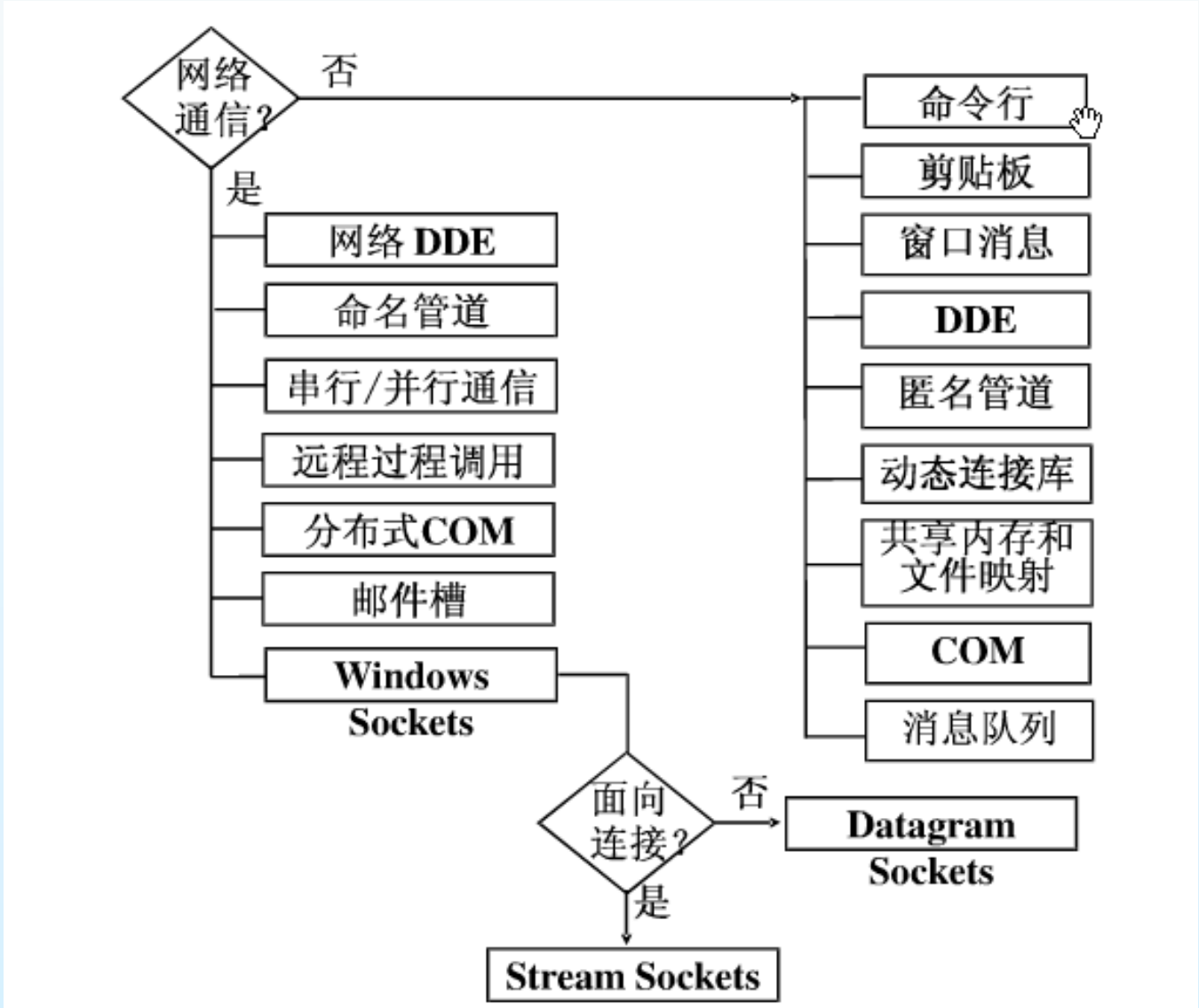
- 套接字(Socket)
- 远程过程调用(RPC)
- 远程方法调用(RMI) (Java)

Sockets

- 套接字被定义为通信的端点
- 套接字由IP地址和端口号连接组成
- **套接字161.25.19.8:1625** 指的是主机**161.25.19.8**上的**1625**端口
- 连接由一对套接字组成



Windows进程间通信



基于文件映射的共享存储区

- 将整个文件映射为进程虚拟地址空间的一部分来访问
  - CreateFileMapping为指定文件创建一个文件映射对象，返回对象指针
  - OpenFileMapping打开一个命名的文件映射对象，返回对象指针
  - MapViewOfFile把文件映射到本进程的地址空间，返回映射地址空间的首地址
- 利用首地址进行读写
  - FlushViewOfFile可把映射地址空间的内容写到物理文件中；
  - UnmapViewOfFile拆除文件映射与本进程地址空间间映射关系；
- 随后，利用CloseHandle关闭文件映射对象

例子-主程序

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include <tchar.h>
#define BUF_SIZE 256
TCHAR szName[]=TEXT("MyFileMappingObject111");
TCHAR szMsg[]=TEXT("Message from first process.");
int _tmain() {
  HANDLE hMapFile;      LPCTSTR pBuf;
    hMapFile = CreateFileMapping(
       INVALID_HANDLE_VALUE,    // use paging file
        NULL,                    // default security
        PAGE_READWRITE,          // read/write access
       0,              // maximum object size (high-order DWORD)
       BUF_SIZE, // maximum object size (low-order DWORD)
       szName);    // name of mapping object
   if(hMapFile == NULL){
              _tprintf(TEXT "Could not create file mapping object (%d). \n, GetLastError());
              return 1;
}
```

例子-主程序

```
pBuf = (LPTSTR) MapViewOfFile(hMapFile,   // handle to map object
                    FILE_MAP_ALL_ACCESS, // read/write permission
                    0,
                    0,
                    BUF_SIZE);
 if(pBuf == NULL){
                _tprintf(TEXT ”Could not map view of file(%d). \n, GetLastError());
                CloseHanle(hMapFile);
                return 1;

 CopyMemory((PVOID)pBuf, szMsg, (_tcslen(szMsg) * sizeof(TCHAR)));

  _getch();
   UnmapViewOfFile(pBuf);
   CloseHandle(hMapFile);
   return 0;
}
```

例子-子程序

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include <tchar.h>
#pragma comment(lib, "user32.lib")

#define BUF_SIZE 256
TCHAR szName[]=TEXT("MyFileMappingObject111");

int _tmain() {
   HANDLE hMapFile;
   LPCTSTR pBuf;
```

例子-子程序

```
hMapFile = OpenFileMapping(
            FILE_MAP_ALL_ACCESS,   // read/write access
            FALSE,                 // do not inherit the name
            szName);               // name of mapping object

if  (hMapFile == NULL){
            _tprintf(TEXT ”Could not open file mapping object (%d). \n,
                        GetLastError());
            return 1;
}
```

例子-子程序

```
pBuf = (LPTSTR) MapViewOfFile(hMapFile, // handle to map object
            FILE_MAP_ALL_ACCESS,  // read/write permission
            0,
            0,
            BUF_SIZE);

if(pBuf == NULL){
            _tprintf(TEXT ”Could not map view of file(%d). \n, GetLastError());
            CloseHanle(hMapFile);
            return 1;
}
```

例子-子程序

```
 MessageBox(NULL, pBuf, TEXT("Process2"), MB_OK);

    UnmapViewOfFile(pBuf);
    CloseHandle(hMapFile);
  return 0;
}
```

剪帖板(Clipboard)

- 当进程间的复杂信息交流需要约定交流信息的格式。剪帖板就是Windows 提供的一种信息交流方式，可增强进程的信息交流能力。
- Windows 提供了一组相关的API来完成应用进程与剪帖板间的格式化信息交流。
- 当执行复制操作时，应用程序将选中的数据以标准的格式或者应用程序定义的格式放到剪贴板中，然后其他的应用程序可以从剪贴板中以其可以支持的格式获取所需要的数据。

剪帖板信息格式

- 剪帖板中提供了许多标准的剪帖板信息格式。如：文本格式和位图格式。
- 允许用户进程注册新的剪帖板信息格式。

Format Type	*Description*
Text Formats	
CF_OEMTEXT	Text containing characters from the OEM character set
CF_TEXT	Text containing characters from the ANSI character set
CF_UNICODETEXT	Text containing Unicode characters
Bitmap formats	
CF_BITMAP	Device-dependent bitmap (HBITMAP)
CF_DIB	Device independent bitmap (HBITMAPINFO)
CF_TIFF	Tagged Image File Format

与剪帖板相关的API

- OpenClipboard: 打开剪帖板；
- CloseClipboard：关闭剪帖板；
- EmptyClipboard：清空剪帖板；
- SetClipboardData：把数据及其格式加入剪帖板；
- GetClipboardData：从剪帖板读取数据；
- RegisterClipboardFormat：注册剪帖板格式

Linux 进程通信

- 主要手段：
  - 管道(Pipe)
  - 信号(Signal)
  - 消息(Message)
  - 共享内存(Shared memory)
  - 信号量(Semaphore)
  - 套接口(Socket)

共享内存

- System V 提供了以下几个函数以实现共享内存：
  - #include <sys/types.h>
  - #include <sys/ipc.h>
  - #include <sys/shm.h>
  - int shmget(key_t key,int size,int shmflg);
  - void *shmat(int shmid,const void *shmaddr,int shmflg);
  - int shmdt(const void *shmaddr);
  - int shmctl(int shmid,int cmd,struct shmid_ds *buf);
- size 是共享内存的大小
- shmat 是用来连接共享内存
- shmdt 是用来断开共享内存的

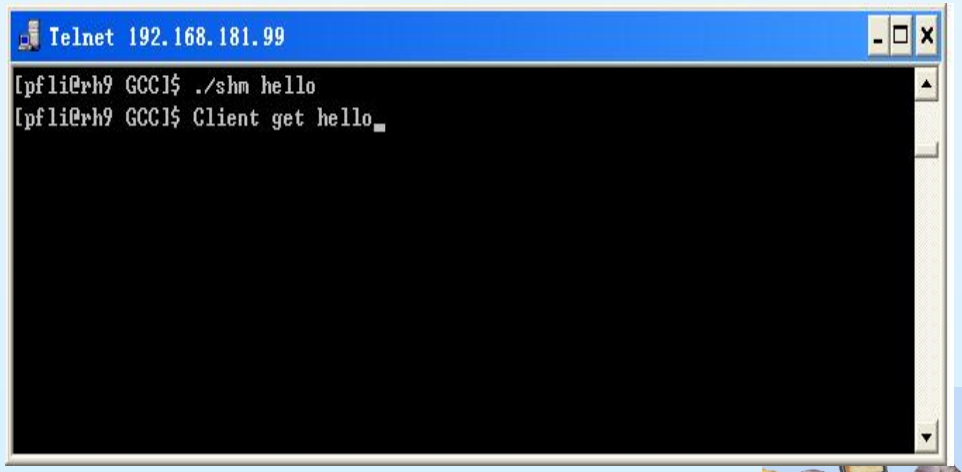
例子

- #include <unistd.h>
- #include <sys/stat.h>
- #include <sys/types.h>
- #include <sys/ipc.h>
- #include <sys/shm.h>
- #define PERM S_IRUSRIS_IWUSR
- int main(int argc,char **argv)
- {
- int shmid;
- char *p_addr,*c_addr;
- if(argc!=2) { printf("Usage：%s\n\a",argv[0]); exit(1); }

- if((shmid=shmget(IPC_PRIVATE,1024,PERM))==-1)
- {
- printf("Create Share Memory Error\n\a");
- exit(1);
- }

例子

- if(fork())
- {
- p_addr=shmat(shmid,0,0);
- memset(p_addr,'\0',1024);
- strncpy(p_addr,argv[1],1024);
- exit(0);
- }
- else
- {
- c_addr=shmat(shmid,0,0);
- printf("Client get %s",c_addr);
- exit(0);
- }
- }



消息队列

- #include <sys/types.h>;
- #include <sys/ipc.h>;
- #include <sys/msg.h>;

- int msgget(key_t key,int msgflg);
- int msgsnd(int msgid,struct msgbuf *msgp,int msgsz,int msgflg);
- int msgrcv(int msgid,struct msgbuf *msgp,int msgsz, long msgtype,int msgflg);
- int msgctl(Int msgid,int cmd,struct msqid_ds *buf);
- struct msgbuf {
- long msgtype; /* 消息类型*/
- ..... /* 其他数据类型*/
- }

server.c

- #include <stdio.h>
- #include <string.h>
- #include <stdlib.h>
- #include <errno.h>
- #include <unistd.h>
- #include <sys/types.h>
- #include <sys/ipc.h>
- #include <sys/stat.h>
- #include <sys/msg.h>

- #define MSG_FILE "server.c"
- #define BUFFER 255
- #define PERM S_IRUSRIS_IWUSR

- struct msgtype {
- long mtype;
- char buffer[BUFFER+1];
- };

server.c

```
int main()
{
struct msgtype msg;  key_t key;  int msgid;

/*key_t 据pathname和proj来创建一个关键字，在创建信号量，创建消息队列
的时候都需要使用。其中pathname必须是一个存在的可访问的路径或文件
，proj必须不得为0。*/
if((key=ftok(MSG_FILE,'a'))==-1)
{
  printf("Creat Key Error\n");
  exit(1);
}

if((msgid=msgget(key,PERM|IPC_CREAT|IPC_EXCL))==-1)
{
  printf("Creat Message Error\n");
  exit(1);
}
```

server.c

```
while(1)
{
   msgrcv(msgid,&msg,sizeof(struct msgtype),1,0);
   printf("Server Receive：%s\n",msg.buffer);
   msg.mtype=2;
   msgsnd(msgid,&msg,sizeof(struct msgtype),0);
}
exit(0);
}
```

client.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/stat.h>

#define MSG_FILE "server.c"
#define BUFFER 255
#define PERM S_IRUSRIS_IWUSR

struct msgtype {
long mtype;
char buffer[BUFFER+1];
};
```

```
int main(int argc,char **argv)
{
struct msgtype msg;
key_t key; int msgid;

if(argc!=2)
{
  printf("Usage：%s string\n\a",argv[0]);
  exit(1);
}
if((key=ftok(MSG_FILE,'a'))==-1)
{
 printf("Creat Key Error：\n");
 exit(1);
}
```

client.c

```
if((msgid=msgget(key,PERM))==-1)
{
  printf("Creat Message Error：\a\n");
  exit(1);
}
msg.mtype=1;
strncpy(msg.buffer,argv[1],BUFFER);
msgsnd(msgid,&msg,sizeof(struct msgtype),0);
memset(&msg,'\0',sizeof(struct msgtype));
msgrcv(msgid,&msg,sizeof(struct msgtype),2,0);
printf("Client receive：%s\n",msg.buffer);
exit(0);
}
```

Telnet 192.168.181.99

```
[pfli@rh9 GCC]$ ./server &
[1] 3709
[pfli@rh9 GCC]$ ./client hello
Server Receive: hello
Client receive: hello
[pfli@rh9 GCC]$ ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 32768      pfli       600        1024       0
0x00000000 65537      pfli       600        1024       0
0x00000000 98306      pfli       600        1024       0

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x6102c2ad 32768      pfli       600        0            0

[pfli@rh9 GCC]$ kill 3709
[pfli@rh9 GCC]$ ipcrm -q 32768
[1]+  Terminated              ./server
[pfli@rh9 GCC]$ ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 32768      pfli       600        1024       0
0x00000000 65537      pfli       600        1024       0
0x00000000 98306      pfli       600        1024       0

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
```