**进程间通信**

一 通信方式

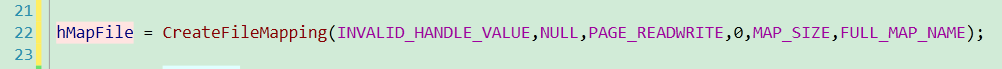
管道，消息队列，共享内存，信号量，信号，套接字

二 应用举例

1. 共享内存通信

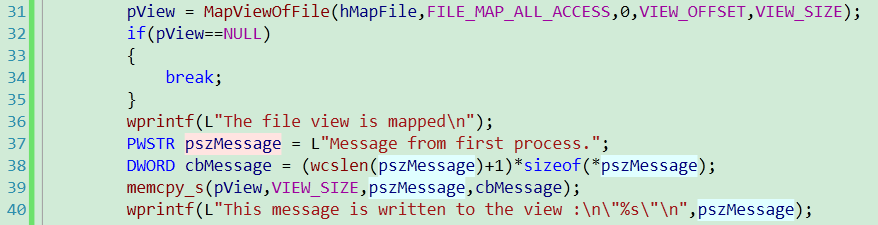
共享内存（也叫内存映射文件）主要是通过映射机制实现的，Windows下进程的地址空间在逻辑上是相互隔离的，但在物理上却是重叠的。所谓的重叠是指同一块内存区域可能被多个进程同时使用。

服务端调用CreateFile创建命名的内存映射文件对象，这时Windows即在物理内存申请一块指定大小的内存区域，返回文件映射对象的句柄hMap。

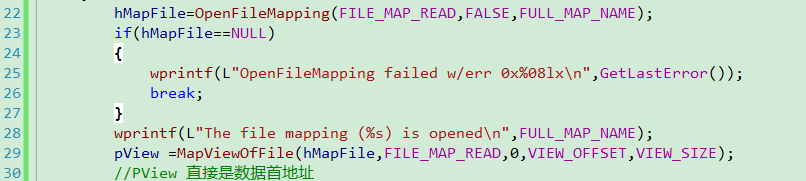


然后通过MapViewOfFile函数，促使Windows将此内存空间映射到进程的地址空间中

，并可以使用memcpy或者strcpy写入数据。

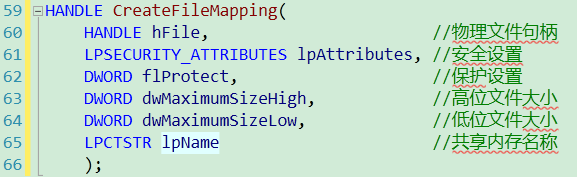


客户端（另一个进程）在访问这块内存时，则必须使用OpenFileMapping函数取得对象句柄hMap,并调用MapViewOfFile函数得到此内存空间的一盒映射，这样就可以把服务端存入的数据给读出来。



1.2 函数

CreateFileMapping



参数：

hFile: 如果创建于物理文件无关的内存映射，可以设置为INVALID\_HANDLE\_VALUE.

lpAttributes: 安全设置，一般设为NULL即可。

flProtect:保护设置 PAGE\_READONLY 或者PAGE\_READWRITE

dwMaximumSizeHigh: 高文件大小。一般设置为0。32位寻址。

dwMaximumSizeLow: 低文件大小。一般填写内存大小

lpName: 共享内存名称，以后指向映射对象以0结尾的字符串。如 “Local\\ SampleMap”

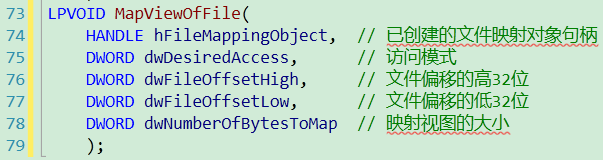
(每个共享内存的名称应该不能相同。)

返回值：

如果函数调用成功，将返回文件映射对象的句柄。  
如果对象在函数调用之前就已存在，函数将返回已存在的对象句柄（size以已存在的对象size为准），这时，GetLastError将返回ERROR\_ALREADY\_EXISTS。  
如果函数失败，返回NULL。可以调用GetLastError获得错误码

MapViewOfFile 获取地址空间指针

进行内存映射文件的读写和一般的文件读写不同，是直接面对你申请的地址空间，为此需要使用MapViewOfFile得到相关地址LPVOID类型的指针。如果需要进行文件写入，可以通过类型转换直接对内存地址进行赋值。如memcpy(lpAddress,lpBuf).



参数：

hFileMappingObject: 由CreateFileMapping或者OpenFileMapping返回文件映像句柄。

dwDesiredAccess: 映像视图访问模式。

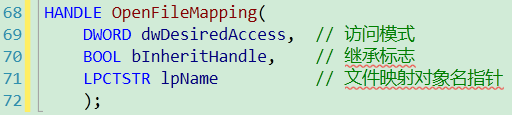
FILE\_MAP\_WRITE:可读写。 保护模式为PAGE\_READWRITE

FILE\_MAP\_READ: 只读视图被创建。保护模式为PAGE\_READONLY

dwFileOffsetLow: 获取的地址相对首地址的偏移量。

dwNumberOfBytesToMap: 映射文件大小，和CreateFileMapping时一样。

OpenFileMapping 打开命名共享内存。



dwDesiredAccess: 访问模式。

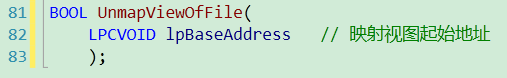
bInheritHandle: 继承标志，是否可以被一个新的进程继承使用，如果为TRUE，就可以被一 个新进程继承句柄。

lpName: 文件映射对象名字串。

返回值：

成功返回一个已命名的文件映射对象，失败返回NULL。

UnmapViewOfFile 卸载内存映射文件地址指针。



lpBaseAddress ：映射视图起始地址，由MapVIewOfFile产生。

返回值：

成功返回非零，失败返回非零。

1. 信号
2. 管道

管道是一种具有两个端点的通信通道，一个管道实际上就是只存在在内存中的文件，对这个文件操作需要两个已经打开文件进行，他们代表管道的两端，也叫两个句柄，管道是一种特殊的文件，不属于一种文件系统，而是一种独立的文件系统，有自己的数据结构，根据管道的使用范围划分为无名管道和命名管道。

无名管道用于父进程和子进程之间，通常父进程创建管道，然后由通信的子

进程继承父进程的读端点句柄和写端点句柄，或者父进程有读写句柄的子进程，这些子进程可以使用管道直接通信，不需要通过父进程。

命名管道，命名管道是为了解决无名管道只能在父子进程间通信而设计的，命名管道是建立在实际的磁盘介质或者文件系统，任何进程可以通过文件名或者路径建立与该文件的联系，命名管道需要一种FIFO文件（先进先出的原则），虽然FIFO文件的inode节点在磁盘上，但仅是一个节点而已，文件的数据还是存在于内存缓冲页面中，和普通管道相同。FIFO文件在磁盘上没有数据块，仅用来标识内核中的一条通道。

命名管道产生FIFO文件，而不产生文件描述符。所以需要调用open（）函数打开。

* 1. 无名管道的创建 pipe（）

企业微信截图_15301819841167

参数返回两个文件描述符，filedis[0] 为读而打开，filedes[1] 为写而打开。

如下例程：

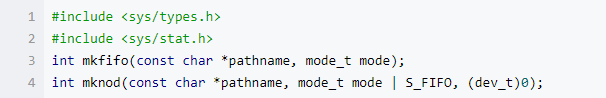
第18行调用pipe()创建管道，得到file\_pipes[2]两个文件描述符。

第29行 子进程调用read() 通过读描述符file\_pipes[0] 读取管道里的数据。

第35行 父进程调用write() 通过写描述符file\_pipes[1] 向管道写入数据。



* 1. 创建有名管道



生成了有名管道以后，就可以使用一般的文件IO函数，如open,close,read,write。

参数：pathname 命名管道名称。（挂载windows文件系统不支持mkfifo）.

Mode\_t 生成管道的权限。（wrx-wrx-wrx,常用0777）.

返回值：成功返回0

一般服务端（生产者或者需要发送消息的一方）这边使用mkfifo()创建FIFO文件。然后调用open（）打开管道文件，获取文件指针，再调用write（）写入数据。

P6X4SD@3N{Q%FJ{(4BTRUPS

LVZ6YJH30OXGR8(APKYOSYFIDSE{0JAJU)6]__6YLA0ETX

而客户端可以直接open（）之后调用read()读取管道文件里面的内容。

LZNKZHQ0N1DY2I066Z`NY)O

IBR3]M0MH@XS~PI}})`(~75

* 1. 函数

//2018.11.27补充linux上常见的两种进程间通信。

1. Sockpair 用于创建一对无名的，相互连接的套接字。
   1. 声明：

#include<sys/types.h>

#include<sys/socket.h>

* 1. 要点

1. Socketpair() 函数用于创建一对无名的，相互连接的套接字。如果函数成功，则返回0， 创建好的套接字分别是sv[0]和sv[1]; 否则返回-1，错误码保存于errno中。
2. 这对套接字可以用于全双工通行，每一个套接字既可以读也可以写。例如，可以往sv[0]中写，从sv[1] 中读；或则从sv[1]中写，从sv[0]中读。
3. 如果往一个套接字（如sv[0]）中写入后，再从该套接字读时会阻塞，只能在另一个套接字中(sv[1])上读成功。
4. 读写操作可以位于同一个进程，也可以分别位于不同的进程，如父子进程。如果时父子进程时，一般会功能分离，一个进程用来读，一个用来写。因为文件描述符sv[0]和sv[1]是进程共享的，所以读的进程要关闭写描述符，反之，写的进程关闭读描述符。
   1. 函数
5. Int socketpair(int domain, int type, int protocol, int sv[]2);

参数：domain， 表示协议族，在linux下只能是AF\_LOCAL或者AF\_UNIX.linux2.6.27后也支 持SOCK\_NONBLOCK和SOCK\_CLOEXEC

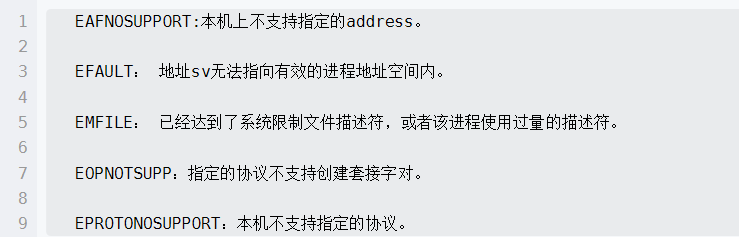
Type, 表示协议类型，SOCK\_STREAM或者SOCK\_DGRAM。基于tcp或者udp.

Protocol, 只能为0。

Sv[2] 套接字句柄，该句柄作用相同，均能进行读写双向操作。

返回结果：0为成功创建，-1为创建失败。并且errno来表明特定的错误号。

一般的错误号：



1. Read
2. write
   1. 使用示例





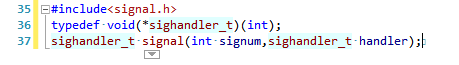
在编译的时候，发现加了-finput参数，便不会有乱码。

1. 信号
   1. 基本定义

#include<signal.h>

* 1. 要点
  2. 函数

1. Sigina()



Signum:信号编号

Handler:处理函数

返回值也是一个sighandler\_t类型的这里返回的是之前的信号处理函数。

信号处理函数是一个带int参数，返回值为void的函数。Handler也可以是两个特殊的值：

SIG\_IGN 屏蔽信号；SIG\_DFL 恢复默认行为。

1. kill() 把信号sig发送给进程pid

~)M0@J$]O{$UHQF~}W[2MHE

Pid 进程id

Sig 信号

* 1. 使用示例