VC 中利用内存映射文件实现进程间通信的方法

田伟，余佥，赵祎骅
（东海舰队参谋部 38 分队，浙江 宁波 315000）

摘　要：利用内存映射文件实现高效的进程间通信，完成实时性要求高的工作。进程使用内存映射文件创建环形缓冲区存取数据达到进程间通信的目的。

关键词：Windows 系统；VC 语言；内存映射文件；事件对象；环形缓冲区
DOI:10.16184/j.cnki.comprg.2017.12.006

1　概述

某工程需要多个进程协同工作，典型应用场景为一个进程向不定数目的其他进程广播变化的态势信息，类似订阅/分发机制，要求简单快捷，程序发布方便，减少依赖库。这就涉及到进程间通信的问题。

因为使用 Win32 下进程间通信方式有多种，需要了解各个方法的特点，找出适合的手段来满足工程的要求。

2　Windows 进程间通信

Windows 下的进程间通信主要有如下几种方法：文件共享、直接读写目标进程内存、动态库共享数据段、内存映射文件、管道、剪贴板、窗口消息、WinSock。下面对主要的几种方法作简要介绍。

2.1　文件共享

数据发送进程打开/创建一个文件并共享，然后向文件中写入数据，数据接收进程打开共享文件，然后读取数据，一般是在循环中读取。通常使用 MFC 封装的 CFile 类实现该功能，示例代码如下：

写入：

```
   CFile file;
   file.Open(...
char buf[100];
   sprintf(buf,"Hello! \r\n",
     file.Write(buf，strlen(buf));
      file.Close();
```

读取：

```
   CFile file;
   file.Open(...
   char Buf[100]={0};
     file.Read(Buf,100);
...
file.Close();
```

2.2　直接读写目标进程内存

该方法就是直接在目标进程分配内存，向该内存区域写数据，通知目标进程读取数据。该方法在实现进程注入等目的时常用。可以调用 Windows API 函数 GlobalAlloc() 或者 VirtualAllocEx() 来分配内存空间，使用内存读写函数 ReadProcessMemory() 和 WriteProcessMemory() 来读写目标进程的内存。

首先打开（或创建，使用 CreateProcess 函数）目标进程：

```
          HANDLE hProc = ::OpenProcess (PROCESS_ALL_ACCESS,FALSE,pid);
```

在目标进程中分配内存：

```
    LPVOID lpMem;
    //分配虚拟内存
       lpMem = ::VirtualAllocEx(hProcess,0, buf_size,MEM_COMMIT,PAGE_READWRITE);
```

写入数据：

```
    //把字符串写入 hProcess 进程的内存
            ::WriteProcessMemory (hProc,lpMem,buf,buf_size,NULL);
```

然后向目标进程发送自定义窗口消息，将内存地址、数据长度传递过去，目的进程响应窗口消息，处理即可。

2.3　动态库共享数据段

该方法是利用动态库中的共享数据段作为加载该动态库的公共存储空间来达到交换数据的目的。示例如下：

```
    //创建动态库时加上共享数据段,还要指定读写和共
//享权限,rws 表示读写共享权限
    #pragma data_seg (".share")
```

收稿日期：2017-03-09

```
int nShare = 0;
#pragma data_seg()
#pragma comment(linker,"/section:.share,rws")
```

在动态库头文件中导出共享的变量：

```
extern int nShare;
```

注意，共享段中的数据必须初始化，否则变量不会放入共享节中。

2.4 内存映射文件

Windows 系统单个计算机上共享数据的底层机制是内存映射文件。同一台计算机上，内存映射文件可达到较高的性能和较小的开销，是进程间通信的较好办法。

多个进程要共享单个文件映射对象必须使用相同的名字来访问该文件映射对象。以下将详细讨论该方法。

2.5 管道方式

管道的类型有两种：匿名管道和命名管道。匿名管道是不命名的，一般用于父进程和子进程之间的通信。命名管道通过一个名字进行标识，使客户端和服务端应用程序可以通过该管道进行通信。

一种简单的匿名管道方式为过滤器模式：一个进程向标准输出（stdout）输出数据，另一个进程通过标准输入（stdin）输入数据，输出进程使用管道操作符“|”连接输入进程。

2.6 剪贴板方式

Windows 剪贴板是一种比较简单同时也是开销比较小的进程间通信手段。其基本原理是由系统预留一块全局共享内存，用来暂存各个进程间进行交换的数据。

在数据放到剪贴板前，首先需要打开剪切板，这里可以用 CWnd 类的 OpenClipboard() 成员函数实现。打开剪切板后，调用 EmptyClipboard() 函数让打开剪切板的当前窗口拥有剪切板。调用 SetClipboardData() 函数向剪切板中放置数据。

2.7 消息方式

Windows 窗口间可以通过发送消息来传递少量的信息，跨进程的消息传递需要向系统注册自定义消息类型，发送至目标窗口后由目标窗口的消息处理函数处理跨进程数据。

2.8 SOCKET 方式

Windows 系统提供了处理网络通信的 Windows API，称之为 Winsock。Winsock 支持多种协议，常用 TCP 和 UDP 两种。使用 Winsock 时，主要有 3 个参数：目的 IP 地址、使用的传输层协议（TCP 或 UDP）和端口号，通过这 3 个参数可唯一确定接收对象。

MFC 提供了 CAsyncSocket 类和 CSocket 类封装了常用的 Winscok API 方便使用，以 UDP 通信为例：

发送方：

```
//初始化 Winsock 环境
AfxSocketInit();
//创建套接字
CSocket sk;
sk.Create(6666/* 端口号 */,SOCK_DGRAM/* 数据报类型 */,NULL);
//发数据
sk.SendTo (“Hello! \r\n”,8,7777/* 目的端口 */,”127.0.0.1”/* 目的地址 */,0);
```

接收方：

```
//初始化环境等
...
//继承 CSocket 类，重载 OnReceive 函数，加上接收处理
Void CMySock::OnReceive(int nErrorCode)
{
    ...
    ReceiveFrom (recBuf,1024, (SOCKADDR*)&ClientAddr,&len,0);
    ...
    CSocket::OnReceive(nErrorCode);
}
```

3 内存映射文件方法具体实现

3.1 内存映射文件类

内存映射文件使用 CreateFileMapping和 MapViewOfFile 两个 Windows API 函数来达成，为方便使用封装为 C++类，采用 VCKBase 网站上介绍的内存映射文件类 CSFMServer/CSFMClient。该类分为服务端和客户端两种，区别为服务端创建内存映射文件并打开；客户端只是根据名称打开现有的映射文件，如果不存在内存映射文件，则 GetBuffer() 方法返回的指针为空。代码如下：

```
//初始化函数中先用客户方式打开内存映射文件，如果存在，则全局写索引使用当前值；如果没有就用服务端方式创建一个并将全局写索引设为 0
psmclnt =new CSFMClient (FILE_MAP_READ | FILE_MAP_WRITE,pMemName);
pShareMem=(BYTE*)psmclnt->GetBuffer();
if(NULL==pShareMem)
{
```

```
    psmsvr =new  CSFMServer (pMemName,pMem-
Name,sizeof(int)+sizeof(CELL)*1000);
    pShareMem=(BYTE*)psmsvr->GetBuffer();
    ZeroMemory(pShareMem,sizeof(int)+sizeof(CELL)
*100);
    delete psmclnt;
    psmclnt=NULL;
    }
```

对内存映射文件的操作通过获得内存块指针并进行读写来完成。

3.2 环形缓冲区类

为有效管理内存映射文件、发送队列和接收队列，定义了环形缓冲区类，因工程需要一写多读，类似于广播发送，故未设置全局读指针。设计思想为在堆上分配（或使用与分配的内存）指定数量和大小的单元构成缓冲区，定义读索引和写索引按照不同的初始化条件，设置了两个构造函数：

```
        //在已分配的内存上创建环形缓冲区,写索引为
//全局,读索引各自保存,读索引初始化为与写索引(全
//局)一致
    CLoopBuffer::CLoopBuffer(void *pBuf,int* pG_In-
dex, int nCellCnt, int nCellSize)
    {
    m_pWriteIndex=pG_Index;
    m_WriteIndex=-1;
    m_ReadIndex=*pG_Index;
    m_nCellCnt=nCellCnt;
    m_nCellSize=nCellSize; //单元大小
    m_pBuf=(BYTE*)pBuf;
    }
      //创建新的环形缓冲区,读写索引均为私有,初始
//化为 0
CLoopBuffer::CLoopBuffer(int nCellCnt, int nCellSize)
    {
    m_pWriteIndex=&m_WriteIndex;
    m_WriteIndex=0;
    m_ReadIndex=0;
    m_nCellCnt=nCellCnt;
    m_nCellSize=nCellSize; //单元大小
    m_pBuf=new BYTE[nCellCnt*nCellSize]; //堆上创
//建缓冲区
    }
    // 读缓冲区
    bool CLoopBuffer::Read(char* pOut)
    {
    if(m_ReadIndex==(*m_pWriteIndex))
        return false;
    else
    {
            memcpy  (pOut,m_pBuf +m_ReadIn-
dex*m_nCellSize,m_nCellSize);
    m_ReadIndex=(++m_ReadIndex)%m_nCellCnt;
        return true;
    }
    }
    // 写缓冲区
    bool CLoopBuffer::Write(char* pIn,int nSize)
    {
    int nIndex=(*m_pWriteIndex)+1;
      if(nIndex==m_ReadIndex)
        return false;
          ZeroMemory(m_pBuf +(*m_pWriteIndex)
*m_nCellSize,m_nCellSize);
      memcpy(m_pBuf +(*m_pWriteIndex)*m_nCell-
Size,pIn,nSize);
        *m_pWriteIndex =(++(*m_pWriteIndex))%
m_nCellCnt;
    }
```

3.3 事件同步对象

Windows 有多种同步对象，使用 MFC 封装的 CMutex、CEvent 对象实现线程同步、资源访问控制等功能。使用 CMutex 类对全局的内存映射文件写操作进行互斥保护，此时需使用有名的对象；使用 CEvent 对象作为消息通知，通知写处理线程、读缓冲线程（跨进程，需使用有名的对象）、读处理线程协同工作，完成写缓存、写入公共区、读公共区并缓存、读处理等工作。

初始化时创建几个事件对象：

```
    ...
    char pEvevntName[31]={0};
    strcpy(pEvevntName,pMemName);
    strcat(pEvevntName,"_ReadEvent");
    char pMutexName[31]={0};
    strcpy(pMutexName,pMemName);
    strcat(pMutexName,"_Mutex");
     pReadEvt =new  CEvent (FALSE,TRUE,pEvevnt-
Name);
    pDataEvt=new CEvent(FALSE,TRUE);
```

(下转第 29 页)

回归测试也可以做成自动化回归测试，就是将发现的bug做成自动化测试用例，在系统进行周期性自动化测试时执行。自动化回归测试可以降低系统测试、维护升级等阶段的成本。

在大数据时代，数字化校园是教育现代化和科技化的发展趋势。数字化校园软件作为数字化校园的重要载体，其质量是关系数据化校园实施的成效。做好数字化校园软件的测试，是关键步骤。测试过程要注重沟通交流，关注客户需求，挖掘需求，做好测试用例，利用自动化测试，做好缺陷管理和回归测试，善于总结归纳，代表客户进行测试，交付高质量的产品。

参考文献

[1] 朱安平. 数字化校园的现实探索.
[2] 张军征. 数字化校园规划与实施.
[3] [美] 梅耶，等. 软件测试的艺术. 张晓明，译.

(上接第26页)

```
pSendDataEvt=new CEvent(FALSE,TRUE);
pMutex=new CMutex(FALSE,pMutexName);
...
```

3.4 编址及寻址

数据交换需要明确源地址和目的地址，为此定义了地址空间及寻址方式。地址定义为双字节无符号短整型，定义65535为广播地址。数据块首4字节为发方地址、收方地址，收到数据时收方判断数据块中目的地址是否为自身或广播地址，符合则处理，否则丢弃。数据块定义：

```
//环形缓冲区单元结构定义
#pragma pack(push)
#pragma pack(1)
typedef struct
{
 USHORT nSrc;
 USHORT nDest;
 USHORT nLen;
 BYTE Buf[1024];
}CELL,*LPCELL;
#pragma pack(pop)
```

3.5 工作线程

为避免堵塞调用者，需要使用单独的线程来处理用户发送数据、接收数据、写公共内存区等功能。工程定义了3个线程:写处理线程、读缓冲线程、读处理线程。用户发送数据时，数据首先压入发送缓冲区，并通知发送线程将数据在适当时候写入内存映射文件；发送线程写入映射文件后通知客户进程中的读线程将映射文件中的数据读入读缓冲区，并通知读处理线程处理数据；读处理线程调用回调函数处理数据。

3.6 动态库封装

使用中将创建/打开内存共享文件、在公共内存区创建环形缓冲区、编址/寻址、访问互斥、无阻塞写入等功能封装入动态库提供用户使用，主要定义并导出了3个接口函数：

```
/* 初始化函数，支持注册多个侦听模块号,psource为本进程模块号数组,nAddrCnt为模块号数量,pMemName为内存映射文件名称,rcvhandler为接收数据处理回调函数 */
 bool WINAPI INIT_SHARE_MEM (int* psource, int nAddrCnt, char*pMemName,PRECEIVEHANDLER rcvhandler);
//清理函数
bool WINAPI FINAL_SHARE_MEM();
//发送数据函数
 bool WINAPI SENDDATA (int source,int dest, BYTE* pBuf,int nLen);
//接收数据处理回调函数定义
 typedef void (WINAPI *PRECEIVEHANDLER)(int source,int dest,BYTE* pBuf,int nLen);
```

4 结语

通过分析了几种主要的Windows系统下进程间通信方式，结合项目实施中的具体需求，使用了内存映射文件来达成高效的数据交换，达到了工程的要求。为了满足用户的需求，将功能封装入动态库中，形成类似软件总线的产品方便使用。目前写操作尚不完善，使用大的缓冲区来减缓读写冲突，需进一步优化。

参考文献

[1] (美) Jeffrey Richter. Windows核心编程. 机械工业出版社.
[2] 侯俊杰. Win32多线程程序设计. 华中科技大学出版社.
[3] (美) Jeff Prosise. MFC Windows程序设计. 2版. 清华大学出版社.
[4] (美) Al Stevens Clayton Walnum. 标准C++宝典. 电子工业出版社.