**Sample 1:** Using Event Objects

<https://msdn.microsoft.com/en-us/library/ms686915(VS.85).aspx#>

**（1）全局变量**

#define THREADCOUNT 4   
HANDLE ghWriteEvent;   
HANDLE ghThreads[THREADCOUNT];  
DWORD WINAPI ThreadProc(LPVOID);

**(2)**[**CreateEvent**](https://msdn.microsoft.com/en-us/library/ms682396(v=vs.85).aspx) function to create a manual-reset event object whose initial state is nonsignaled

// Create a manual-reset event object. The write thread sets this  
 // object to the signaled state when it finishes writing to a   
 // shared buffer.   
  
 ghWriteEvent = CreateEvent(   
 NULL, // default security attributes  
 TRUE, // manual-reset event  
 FALSE, // initial state is nonsignaled  
 TEXT("WriteEvent") // object name  
 );

(3)in Master WriteToBufffer thread--->Set ghWriteEvent to signaled

SetEvent(ghWriteEvent)

(4)in each thread function, there is one statement to wait for ghWriteEvent to become signaled

ghThreads[i] = CreateThread(  
 NULL, // default security  
 0, // default stack size  
 ThreadProc, // name of the thread function  
 NULL, // no thread parameters  
 0, // default startup flags  
 &dwThreadID);

in ThreadProc, it includes:

dwWaitResult = WaitForSingleObject(   
 ghWriteEvent, // event handle  
 INFINITE); // indefinite wait

(5)WaitForMultipleObjects to check if all the threads are completed.

// The handle for each thread is signaled when the thread is  
 // terminated.  
 dwWaitResult = WaitForMultipleObjects(  
 THREADCOUNT, // number of handles in array  
 ghThreads, // array of thread handles  
 TRUE, // wait until all are signaled  
 INFINITE);

**Sample 2:**

this is a best one to explain the difference between critical section, mutex, semaphore, event

<http://www.codeproject.com/Articles/7953/Thread-Synchronization-for-Beginners>

**Sample 3:**

<http://www.cplusplus.com/forum/unices/23437/>

**Sample 4:**

<http://www.codeguru.com/cpp/w-p/system/sharedmemory/article.php/c2879/Shared-Memory-Inter-Process-Communication-IPC.htm>

**Sample 5:**

[C++使用共享内存实现进程间通信](http://blog.csdn.net/zy_dreamer/article/details/8877376)《--------------比较容易理解 read first

<http://blog.csdn.net/zy_dreamer/article/details/8877376>

文件映射是一种实现进程间单向或双向通信的机制。它允许两个或多个本地进程间相互通信。为了共享文件或内存，所有的进程必须使用相同的文件映射的名字或是句柄[这里用FULL\_MAP\_NAME]。

为了实现共享文件，第一个进程先调用CreateFile方法。接下来调用CreateFileMapping方法来创建一个文件映射对象。并为文件映射指明一个句柄和名称。由于事件，信号，互斥对象和文件映射等这些内核对象都共享同一个名字空间，所以如果这个名字和其他一个对象的名称重名的话那么将创建失败。

为了实现共享内存，进程应首先调用CreateFileMapping函数然后在hFile参数中传入INVALID\_HANDLE\_VALUE宏来替代句柄。相应的文件映射对象会从系统的分页文件中获得一段内存。如果hFile参数的值是INVALID\_HANDLE\_VALUE，那么你在调用CreateFileMapping时必须给共享内存指定一个大小值。

使用共享内存或文件的进程必须使用MapViewOfFile函数或MapViewOfFileEx函数来创建一个文件视图。

下面我们创建一个名称为"Local\SampleMap"的文件映射对象，并将一个字符串写入到文件映射中。

我们将创建两个程序，一个是服务sever程序，一个是客户client程序。

服务程序负责创建文件映射。

服务程序命名为CppFileMappingServer，它的执行过程是

1.CreateFileMapping创建一个特定大小的文件映射对象，名称为“Local\SampleMap”

2.将这个对象的文件视图映射MapViewOfFile到进程的地址空间，然后向视图中写入字符串memcpy\_s(pView, VIEW\_SIZE, pszMessage, cbMessage);。

接下来执行客户程序CppFileMappingClient，它首先打开OpenFileMapping这个名称为“Local\SampleMap”的文件映射对象。然后把相同的文件映射MapViewOfFile视图映射到自己的地址空间中。然后从视图中读取服务进程所写入的数据wprintf(L"Read from the file mapping:\n\"%s\"\n", (**PWSTR**)pView); 。

**Sample 6:**

<http://my.oschina.net/u/131904/blog/133175>

　　　进程通常被定义为一个正在运行的程序的实例， 为了能在两个进程之间进行通讯，我们可以通过下面的方法来实现：

2。使用共享内存方式（Shared Memory)

a.设定一块共享内存区域

HANDLE CreateFileMapping(HANDLE,LPSECURITY\_ATTRIBUTES, DWORD, DWORD, DWORD, LPCSTR)

产生一个file-mapping核心对象

LPVOID MapViewOfFile(

HANDLE hFileMappingObject,

DWORD dwDesiredAcess,

DWORD dwFileOffsetHigh,

DWORD dwFileOffsetLow,

DWORD dwNumberOfBytesToMap

);

得到共享内存的指针

b.找出共享内存

决定这块内存要以点对点（peer to peer)的形式呈现

每个进程都必须有相同的能力，产生共享内存并将它初始化。每个进程

都应该调用CreateFileMapping(),然后调用GetLastError().如果传回的

错误代码是ERROR\_ALREADY\_EXISTS,那么进程就可以假设这一共享内存区域已经被别的进程打开并初始化了，否则该进程就可以合理的认为自己 排在第 一位，并接下来将共享内存初始化。

还是要使用client/server架构中

只有server进程才应该产生并初始化共享内存。所有的进程都应该使用

HANDLE OpenFileMapping(DWORD dwDesiredAccess,

BOOL bInheritHandle,

LPCTSTR lpName);

再调用MapViewOfFile(),取得共享内存的指针

c.同步处理(Mutex)

d.清理(Cleaning up) BOOL UnmapViewOfFile(LPCVOID lpBaseAddress);

CloseHandle()

这个创建内存共享我觉得是最简单的方法：

//共内存句柄

HANDLE m\_hwdShareMemery;

//共享内存映射

char\* m\_strBufStart;

void OnBnClickedBtnCreate()

{

m\_hwdShareMemery = CreateFileMapping(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, 0, 2048, \_T("TestMem"));

if (m\_hwdShareMemery == NULL)

{

AfxMessageBox(\_T("Create Share Memery Failed !"));

}

}

void OnBnClickedBtnMap()

{

HANDLE handleFile = NULL;

handleFile = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, false, \_T("TestMem"));

m\_strBufStart = (char\*)MapViewOfFile(handleFile, FILE\_MAP\_ALL\_ACCESS, 0, 0, 2048);

}

void OnBnClickedBtnWrite()

{

CString str;

m\_edtWrite.GetWindowTextW(str);

int nStrLen = str.GetLength();

sprintf\_s(m\_strBufStart,nStrLen\*2, "%S", str);

m\_edtWrite.SetWindowTextW(\_T(""));

}

void OnBnClickedBtnRead()

{

CString strRead(m\_strBufStart);

m\_edtRead.SetWindowTextW(strRead);

}

**Sample 7: Using mutex Objects**

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms686927(v=vs.85).aspx>

in main thread:

ghMutex = CreateMutex(   
 NULL, // default security attributes  
 FALSE, // initially not owned  
 NULL); // unnamed mutex

in thread function includes two calls:

1. WaitForSingleObject(ghMutex, INFINITE);
2. ReleaseMutex(ghMutex);