**[win32下进程间通信——共享内存](http://blog.csdn.net/wcyoot/article/details/7363393)**

分类： [Windows核心编程](http://blog.csdn.net/sszgg2006/article/category/1312047)2013-02-06 10:082614人阅读[评论](http://blog.csdn.net/sszgg2006/article/details/8573348" \l "comments)(0)[收藏](javascript:void(0);)[举报](http://blog.csdn.net/sszgg2006/article/details/8573348#report)

**一、引言**  
在Windows程序中，各个进程之间常常需要交换数据，进行数据通讯。WIN32 API提供了许多函数使我们能够方便高效的进行进程间的通讯，通过这些函数我们可以控制不同进程间的数据交换.  
进程间通讯(即：同机通讯)和数据交换有多种方式：消息、共享内存、匿名（命名）管道、邮槽、Windows套接字等多种技术。“共享内存”（shared memory）可以定义为对一个以上的进程是可见的内存或存在于多个进程的虚拟地址空间。例如：如果两个进程使用相同的DLL，只把DLL的代码页装入内存一次，其他所有映射这个DLL的进程只要共享这些代码页就可以了；利用消息机制实现IPC虽然有交换的数据量小、携带的信息少等缺点，但由于其实现方便、应用灵活而广泛应用于无须大量、频繁数据交换的内部进程通讯系统之中。  
**二、同机进程间共享内存的实现**  
采用内存映射文件实现WIN32进程间的通讯：Windows中的内存映射文件的机制为我们高效地操作文件提供了一种途径，它允许我们在WIN32进程中保留一段内存区域，把硬盘或页文件上的目标文件映射到这段虚拟内存中。注意：在程序实现中必须考虑各进程之间的同步问题。

具体实现步骤如下：  
1、在服务器端进程中调用内存映射API函数CreateFileMapping创建一个有名字标识的共享内存；  
函数CreateFileMapping原型  
HANDLE CreateFileMapping (  
HANDLE hFile, // 映射文件的句柄，若设为0xFFFFFFFF(即：INVALID\_HANDLE\_VALUE)则创建一个进程间共享的对象  
LPSECURITY\_ATTRIBUTES lpFileMappingAttributes, //安全属性

DWORD flProtect, //保护方式  
DWORD dwMaximumSizeHigh, //对象的大小  
DWORD dwMaximumSizeLow,   
LPCTSTR lpName // 映射文件名，即共享内存的名称 );  
与虚拟内存类似，保护方式参数可以是PAGE\_READONLY或是PAGE\_READWRITE。如果多进程都对同一共享内存进行写访问，则必须保持相互间同步。映射文件还可以指定PAGE\_WRITECOPY标志，可以保证其原始数据不会遭到破坏，同时允许其他进程在必要时自由的操作数据的拷贝。  
例如：创建一个名为“zzj”的长度为4096字节的有名映射文件：  
HANDLE m\_hMapFile=CreateFileMapping((HANDLE)0xFFFFFFFF)，  
NULL，PAGE\_READWRITE，0，0x1000，" zzj")；  
2、在创建文件映射对象后，服务器端进程调用MapViewOfFile函数映射到本进程的地址空间内；  
例：映射缓存区视图  
void\* m\_pBaseMapFile=MapViewOfFile(m\_hMapFile，  
FILE\_MAP\_READ|FILE\_MAP\_WRITE，0，0，0);  
3、客户端进程访问共享内存对象，需要通过内存对象名调用OpenFileMapping函数，以获得共享内存对象的句柄  
HANDLE m\_hMapFile =OpenFileMapping(FILE\_MAP\_WRITE，FALSE，" zzj");   
4、如果客户端进程获得共享内存对象的句柄成功，则调用MapViewOfFile函数来映射对象视图。用户可以使用该对象视图来进行数据读写操作，以达到数据通讯的目的。  
例：映射缓存区视图  
void\* m\_pBaseMapFile=MapViewOfFile(m\_hMapFile，FILE\_MAP\_READ|FILE\_MAP\_WRITE，0，0，0);  
5、当用户进程结束使用共享内存后，调用UnmapViewOfFile函数以取消其地址空间内的视图：  
if (m\_pBaseMapFile)   
{   
UnmapViewOfFile(m\_pBaseMapFile);   
SharedMapView=NULL;   
}

三、使用文件映射实现共享内存。

FileMapping用于将存在于磁盘的文件放进一个进程的虚拟地址空间，并在该进程的虚拟地址空间中产生一个区域用于“存放”该文件，这个空间就叫做File View（存放在进程的虚拟内存中），系统并同时产生一个File Mapping Object（存放于物理内存中）用于维持这种映射关系，这样当多个进程需要读写那个文件的数据时，它们的File View其实对应的都是同一个File Mapping Object，这样做可节省内存和保持数据的同步性，并达到数据共享的目的。

当然在一个应用向文件中写入数据时，其它进程不应该去读取这个正在写入的数据。这就需要进行一些同步的操作。下边来看一下具体的API。  
CreateFileMaping的用法：  
HANDLE CreateFileMapping( //返回FileMapping Object的句柄  
HANDLE hFile, //想要产生映射的文件的句柄  
LPSECURITY\_ATTRIBUTES lpAttributes, //安全属性（只对NT和2000生效）  
DWORD flProtect, //保护标致  
DWORD dwMaximumSizeHigh, //在DWORD的高位中存放

File Mapping Object //的大小  
DWORD dwMaximumSizeLow, //在DWORD的低位中存放

File Mapping Object //的大小（通常这两个参数有一个为0）  
LPCTSTR lpName //File Mapping Object的名称。  
);

1)物理文件句柄  
任何可以获得的物理文件句柄，如果你需要创建一个物理文件无关的内存映射也无妨，将它设置成为 0xFFFFFFFF(INVALID\_HANDLE\_VALUE)就可以了.

如果需要和物理文件关联，要确保你的物理文件创建的时候的访问模式和"保护设置"匹配，比如:物理文件只读，内存映射需要读写就会发生错误。推荐你的物理文件使用独占方式创建。

如果使用 INVALID\_HANDLE\_VALUE，也需要设置需要申请的内存空间的大小，无论物理文件句柄参数是否有效,这样CreateFileMapping就可以创建一个和物理文件大小无关的内存空间给你,甚至超过实际文件大小，如果你的物理文件有效，而大小参数为0，则返回给你的是一个和物理文件大小一样的内存空间地址范围。返回给你的文件映射地址空间是可以通过复制，集成或者命名得到，初始内容为0。

2)保护设置  
就是安全设置,不过一般设置NULL就可以了,使用默认的安全配置.在win2k下如果需要进行限制,这是针对那些将内存文件映射共享给整个网络上面的应用进程使用是,可以考虑进行限制.

3)高位文件大小  
32位地址空间,设置为0。  
4) 共享内存名称

命名可以包含 "Global"或者 "Local"前缀在全局或者会话名空间初级文件映射.其他部分可以包含任何除了()以外的字符,可以参考Kernel Object Name Spaces.

5)调用CreateFileMapping的时候GetLastError的对应错误  
ERROR\_FILE\_INVALID 如果企图创建一个零长度的文件映射,应有此报  
ERROR\_INVALID\_HANDLE 如果发现你的命名内存空间和现有的内存映射,互斥量,信号量,临界区同名就麻烦了  
ERROR\_ALREADY\_EXISTS 表示内存空间命名已经存在

使用函数CreateFileMapping创建一个想共享的文件数据句柄，然后使用MapViewOfFile来获取共享的内存地址，然后使用OpenFileMapping函数在另一个进程里打开共享文件的名称，这样就可以实现不同的进程共享数据。

代码示例：这个程序包括一个客户端和一个服务端，服务端创建共享内存，客户端打开共享内存，两者通过两个事件互斥访问共享内存，实现一个小功能，就是服务端进程从控制台读入数据发送给客户端进程。

服务端：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/sszgg2006/article/details/8573348)

1. #include "stdafx.h"
2. #include <Windows.h>
3. #include <iostream>
4. using namespace std;
5. **int** main()
6. {
7. **HANDLE** hMutex = NULL;
8. **HANDLE** hFileMapping = NULL;
9. **LPVOID** lpShareMemory = NULL;
10. **HANDLE** hServerWriteOver = NULL;
11. **HANDLE** hClientReadOver = NULL;
12. //create share memory
13. hFileMapping = CreateFileMapping(INVALID\_HANDLE\_VALUE,
14. NULL,
15. PAGE\_READWRITE,
16. 0,
17. 1024\*1024,
18. L"ShareMemoryTest");
19. if (NULL == hFileMapping)
20. {
21. cout << "CreateFileMapping fail:" << GetLastError() << endl;
22. goto SERVER\_SHARE\_MEMORY\_END;
23. }
24. lpShareMemory = MapViewOfFile(hFileMapping,
25. FILE\_MAP\_ALL\_ACCESS,
26. 0,
27. 0, //memory start address
28. 0); //all memory space
29. if (NULL == lpShareMemory)
30. {
31. cout << "MapViewOfFile" << GetLastError() << endl;
32. goto SERVER\_SHARE\_MEMORY\_END;
33. }
34. //373
35. hMutex = CreateMutex(NULL, FALSE, L"SM\_Mutex");
36. if (NULL == hMutex || ERROR\_ALREADY\_EXISTS == GetLastError())
37. {
38. cout << "CreateMutex" << GetLastError() << endl;
39. goto SERVER\_SHARE\_MEMORY\_END;
40. }//多个线程互斥访问
41. //send data
42. hServerWriteOver = CreateEvent(NULL,
43. TRUE,
44. FALSE,
45. L"ServerWriteOver");
46. hClientReadOver = CreateEvent(NULL,
47. TRUE,
48. FALSE,
49. L"ClientReadOver");
50. if (NULL == hServerWriteOver ||
51. NULL == hClientReadOver)
52. {
53. cout << "CreateEvent" << GetLastError() << endl;
54. goto SERVER\_SHARE\_MEMORY\_END;
55. }
56. **char** p = 0;
57. **char**\* q = (**char**\*)lpShareMemory;
58. do
59. {
60. p = getchar();
61. if (WaitForSingleObject(hClientReadOver, 5\*1000) != WAIT\_OBJECT\_0)
62. goto SERVER\_SHARE\_MEMORY\_END;
63. q[0] = p;
64. if (!ResetEvent(hClientReadOver)) goto SERVER\_SHARE\_MEMORY\_END;//把指定的事件对象设置为无信号状态
65. if (!SetEvent(hServerWriteOver)) goto SERVER\_SHARE\_MEMORY\_END;//把指定的事件对象设置为有信号状态
66. } while (p != '\n');
67. SERVER\_SHARE\_MEMORY\_END:
68. //release share memory
69. if (NULL != hServerWriteOver) CloseHandle(hServerWriteOver);
70. if (NULL != hClientReadOver) CloseHandle(hClientReadOver);
71. if (NULL != lpShareMemory) UnmapViewOfFile(lpShareMemory);
72. if (NULL != hFileMapping) CloseHandle(hFileMapping);
73. if (NULL != hMutex) ReleaseMutex(hMutex);
74. return 0;
75. }

#include "stdafx.h"

#include <Windows.h>

#include <iostream>

using namespace std;

int main()

{

HANDLE hMutex = NULL;

HANDLE hFileMapping = NULL;

LPVOID lpShareMemory = NULL;

HANDLE hServerWriteOver = NULL;

HANDLE hClientReadOver = NULL;

//create share memory

hFileMapping = CreateFileMapping(INVALID\_HANDLE\_VALUE,

NULL,

PAGE\_READWRITE,

0,

1024\*1024,

L"ShareMemoryTest");

if (NULL == hFileMapping)

{

cout << "CreateFileMapping fail:" << GetLastError() << endl;

goto SERVER\_SHARE\_MEMORY\_END;

}

lpShareMemory = MapViewOfFile(hFileMapping,

FILE\_MAP\_ALL\_ACCESS,

0,

0, //memory start address

0); //all memory space

if (NULL == lpShareMemory)

{

cout << "MapViewOfFile" << GetLastError() << endl;

goto SERVER\_SHARE\_MEMORY\_END;

}

//373

hMutex = CreateMutex(NULL, FALSE, L"SM\_Mutex");

if (NULL == hMutex || ERROR\_ALREADY\_EXISTS == GetLastError())

{

cout << "CreateMutex" << GetLastError() << endl;

goto SERVER\_SHARE\_MEMORY\_END;

}//多个线程互斥访问

//send data

hServerWriteOver = CreateEvent(NULL,

TRUE,

FALSE,

L"ServerWriteOver");

hClientReadOver = CreateEvent(NULL,

TRUE,

FALSE,

L"ClientReadOver");

if (NULL == hServerWriteOver ||

NULL == hClientReadOver)

{

cout << "CreateEvent" << GetLastError() << endl;

goto SERVER\_SHARE\_MEMORY\_END;

}

char p = 0;

char\* q = (char\*)lpShareMemory;

do

{

p = getchar();

if (WaitForSingleObject(hClientReadOver, 5\*1000) != WAIT\_OBJECT\_0)

goto SERVER\_SHARE\_MEMORY\_END;

q[0] = p;

if (!ResetEvent(hClientReadOver)) goto SERVER\_SHARE\_MEMORY\_END;//把指定的事件对象设置为无信号状态

if (!SetEvent(hServerWriteOver)) goto SERVER\_SHARE\_MEMORY\_END;//把指定的事件对象设置为有信号状态

} while (p != '\n');

SERVER\_SHARE\_MEMORY\_END:

//release share memory

if (NULL != hServerWriteOver) CloseHandle(hServerWriteOver);

if (NULL != hClientReadOver) CloseHandle(hClientReadOver);

if (NULL != lpShareMemory) UnmapViewOfFile(lpShareMemory);

if (NULL != hFileMapping) CloseHandle(hFileMapping);

if (NULL != hMutex) ReleaseMutex(hMutex);

return 0;

}

客户端：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/sszgg2006/article/details/8573348)

1. #include "stdafx.h"
2. #include <Windows.h>
3. #include <iostream>
4. using namespace std;
5. **int** main()
6. {
7. **HANDLE** hMutex = NULL;
8. **HANDLE** hFileMapping = NULL;
9. **LPVOID** lpShareMemory = NULL;
10. **HANDLE** hServerWriteOver = NULL;
11. **HANDLE** hClientReadOver = NULL;
12. hMutex = OpenMutex(MUTEX\_ALL\_ACCESS,
13. FALSE,
14. L"SM\_Mutex");
15. if (NULL == hMutex)
16. {
17. if (ERROR\_FILE\_NOT\_FOUND == GetLastError())
18. {
19. cout << "OpenMutex fail: file not found!" << endl;
20. goto CLIENT\_SHARE\_MEMORY\_END;
21. }
22. else
23. {
24. cout << "OpenMutex fail:" << GetLastError() << endl;
25. goto CLIENT\_SHARE\_MEMORY\_END;
26. }
27. }
28. if (WaitForSingleObject(hMutex, 5000) != WAIT\_OBJECT\_0)//hMutex 一旦互斥对象处于有信号状态，则该函数返回
29. {
30. **DWORD** dwErr = GetLastError();
31. goto CLIENT\_SHARE\_MEMORY\_END;
32. }
33. //open share memory
34. hFileMapping = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS,
35. FALSE,
36. L"ShareMemoryTest");
37. if (NULL == hFileMapping)
38. {
39. cout << "OpenFileMapping" << GetLastError() << endl;
40. goto CLIENT\_SHARE\_MEMORY\_END;
41. }
42. lpShareMemory = MapViewOfFile(hFileMapping,
43. FILE\_MAP\_ALL\_ACCESS,
44. 0,
45. 0,
46. 0);
47. if (NULL == lpShareMemory)
48. {
49. cout << "MapViewOfFile" << GetLastError() << endl;
50. goto CLIENT\_SHARE\_MEMORY\_END;
51. }
52. //read and write data
53. hServerWriteOver = CreateEvent(NULL,
54. TRUE,
55. FALSE,
56. L"ServerWriteOver");
57. hClientReadOver = CreateEvent(NULL,
58. TRUE,
59. FALSE,
60. L"ClientReadOver");
61. if (NULL == hServerWriteOver ||
62. NULL == hClientReadOver)
63. {
64. cout << "CreateEvent" << GetLastError() << endl;
65. goto CLIENT\_SHARE\_MEMORY\_END;
66. }
67. **char** p = 0;
68. **char**\* q = (**char**\*)lpShareMemory;
69. do
70. {
71. if (!SetEvent(hClientReadOver))
72. goto CLIENT\_SHARE\_MEMORY\_END;
73. if (WaitForSingleObject(hServerWriteOver, INFINITE) != WAIT\_OBJECT\_0)
74. goto CLIENT\_SHARE\_MEMORY\_END;
75. p = q[0];
76. putchar(p);
77. if (!ResetEvent(hServerWriteOver))
78. goto CLIENT\_SHARE\_MEMORY\_END;
79. } while (p != '\n');
80. CLIENT\_SHARE\_MEMORY\_END:
81. //release share memory
82. if (NULL != hServerWriteOver) CloseHandle(hServerWriteOver);
83. if (NULL != hClientReadOver) CloseHandle(hClientReadOver);
84. if (NULL != lpShareMemory) UnmapViewOfFile(lpShareMemory);
85. if (NULL != hFileMapping) CloseHandle(hFileMapping);
86. if (NULL != hMutex) ReleaseMutex(hMutex);
87. return 0;
88. }

#include "stdafx.h"

#include <Windows.h>

#include <iostream>

using namespace std;

int main()

{

HANDLE hMutex = NULL;

HANDLE hFileMapping = NULL;

LPVOID lpShareMemory = NULL;

HANDLE hServerWriteOver = NULL;

HANDLE hClientReadOver = NULL;

hMutex = OpenMutex(MUTEX\_ALL\_ACCESS,

FALSE,

L"SM\_Mutex");

if (NULL == hMutex)

{

if (ERROR\_FILE\_NOT\_FOUND == GetLastError())

{

cout << "OpenMutex fail: file not found!" << endl;

goto CLIENT\_SHARE\_MEMORY\_END;

}

else

{

cout << "OpenMutex fail:" << GetLastError() << endl;

goto CLIENT\_SHARE\_MEMORY\_END;

}

}

if (WaitForSingleObject(hMutex, 5000) != WAIT\_OBJECT\_0)//hMutex 一旦互斥对象处于有信号状态，则该函数返回

{

DWORD dwErr = GetLastError();

goto CLIENT\_SHARE\_MEMORY\_END;

}

//open share memory

hFileMapping = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS,

FALSE,

L"ShareMemoryTest");

if (NULL == hFileMapping)

{

cout << "OpenFileMapping" << GetLastError() << endl;

goto CLIENT\_SHARE\_MEMORY\_END;

}

lpShareMemory = MapViewOfFile(hFileMapping,

FILE\_MAP\_ALL\_ACCESS,

0,

0,

0);

if (NULL == lpShareMemory)

{

cout << "MapViewOfFile" << GetLastError() << endl;

goto CLIENT\_SHARE\_MEMORY\_END;

}

//read and write data

hServerWriteOver = CreateEvent(NULL,

TRUE,

FALSE,

L"ServerWriteOver");

hClientReadOver = CreateEvent(NULL,

TRUE,

FALSE,

L"ClientReadOver");

if (NULL == hServerWriteOver ||

NULL == hClientReadOver)

{

cout << "CreateEvent" << GetLastError() << endl;

goto CLIENT\_SHARE\_MEMORY\_END;

}

char p = 0;

char\* q = (char\*)lpShareMemory;

do

{

if (!SetEvent(hClientReadOver))

goto CLIENT\_SHARE\_MEMORY\_END;

if (WaitForSingleObject(hServerWriteOver, INFINITE) != WAIT\_OBJECT\_0)

goto CLIENT\_SHARE\_MEMORY\_END;

p = q[0];

putchar(p);

if (!ResetEvent(hServerWriteOver))

goto CLIENT\_SHARE\_MEMORY\_END;

} while (p != '\n');

CLIENT\_SHARE\_MEMORY\_END:

//release share memory

if (NULL != hServerWriteOver) CloseHandle(hServerWriteOver);

if (NULL != hClientReadOver) CloseHandle(hClientReadOver);

if (NULL != lpShareMemory) UnmapViewOfFile(lpShareMemory);

if (NULL != hFileMapping) CloseHandle(hFileMapping);

if (NULL != hMutex) ReleaseMutex(hMutex);

return 0;

}