                            \_\_leave;  
                    }  
                    newAceIndex++;  
            }  
        }  
  
        //   
        // STEP 14: 把一个 access-allowed 的ACE 加入到新的DACL中  
        //     前面的循环拷贝了所有的非继承且SID为其它用户的ACE，退出循环的第一件事  
        // 就是加入我们指定的ACE。请注意首先先动态装载了一个AddAccessAllowedAceEx  
        // 的API函数，如果装载不成功，就调用AddAccessAllowedAce函数。前一个函数仅  
        // 在Windows 2000以后的版本支持，NT则没有，我们为了使用新版本的函数，我们首  
        // 先先检查一下当前系统中可不可以装载这个函数，如果可以则就使用。使用动态链接  
        // 比使用静态链接的好处是，程序运行时不会因为没有这个API函数而报错。  
        //   
        // Ex版的函数多出了一个参数AceFlag（第三人参数），用这个参数我们可以来设置一  
        // 个叫ACE\_HEADER的结构，以便让我们所设置的ACE可以被其子目录所继承下去，而   
        // AddAccessAllowedAce函数不能定制这个参数，在AddAccessAllowedAce函数  
        // 中，其会把ACE\_HEADER这个结构设置成非继承的。  
        //   
        \_AddAccessAllowedAceEx = (AddAccessAllowedAceExFnPtr)  
            GetProcAddress(GetModuleHandle(TEXT("advapi32.dll")),  
            "AddAccessAllowedAceEx");  
        if (\_AddAccessAllowedAceEx) {  
            if (!\_AddAccessAllowedAceEx(pNewACL, ACL\_REVISION2,  
                CONTAINER\_INHERIT\_ACE | OBJECT\_INHERIT\_ACE ,  
                dwAccessMask, pUserSID)) {  
                    \_tprintf(TEXT("AddAccessAllowedAceEx() failed. Error %d\n"),  
                        GetLastError());  
                    \_\_leave;  
            }  
        }else{  
            if (!AddAccessAllowedAce(pNewACL, ACL\_REVISION2,   
                dwAccessMask, pUserSID)) {  
                    \_tprintf(TEXT("AddAccessAllowedAce() failed. Error %d\n"),  
                        GetLastError());  
                    \_\_leave;  
            }  
        }  
        //   
        // STEP 15: 按照已存在的ACE的顺序拷贝从父目录继承而来的ACE  
        //   
        bitset<32> bit(dwAccessMask);  
        if (fDaclPresent && AclInfo.AceCount) {  
            for (;   
                CurrentAceIndex < AclInfo.AceCount;  
                CurrentAceIndex++) {  
                    //   
                    // STEP 16: 从文件（目录）的DACL中继续取ACE  
                    //   
                    if (!GetAce(pACL, CurrentAceIndex, &pTempAce)) {  
                        \_tprintf(TEXT("GetAce() failed. Error %d\n"),   
                            GetLastError());  
                        \_\_leave;  
                    }  
                    //   
                    // STEP 17: 把ACE加入到新的DACL中  
                    //   
                    if (!AddAce(pNewACL, ACL\_REVISION, MAXDWORD, pTempAce,  
                        ((PACE\_HEADER) pTempAce)->AceSize)) {  
                            \_tprintf(TEXT("AddAce() failed. Error %d\n"),   
                                GetLastError());  
                            \_\_leave;  
                    }  
            }  
        }  
        //   
        // STEP 18: 把新的ACL设置到新的SD中  
        //   
        if (!SetSecurityDescriptorDacl(&newSD, TRUE, pNewACL,   
            FALSE)) {  
                \_tprintf(TEXT("SetSecurityDescriptorDacl() failed. Error %d\n"),  
                    GetLastError());  
                \_\_leave;  
        }  
        //   
        // STEP 19: 把老的SD中的控制标记再拷贝到新的SD中，我们使用的是一个叫   
        // SetSecurityDescriptorControl() 的API函数，这个函数同样只存在于  
        // Windows 2000以后的版本中，所以我们还是要动态地把其从advapi32.dll   
        // 中载入，如果系统不支持这个函数，那就不拷贝老的SD的控制标记了。  
        //   
        \_SetSecurityDescriptorControl =(SetSecurityDescriptorControlFnPtr)  
            GetProcAddress(GetModuleHandle(TEXT("advapi32.dll")),  
            "SetSecurityDescriptorControl");  
        if (\_SetSecurityDescriptorControl) {  
            SECURITY\_DESCRIPTOR\_CONTROL controlBitsOfInterest = 0;  
            SECURITY\_DESCRIPTOR\_CONTROL controlBitsToSet = 0;  
            SECURITY\_DESCRIPTOR\_CONTROL oldControlBits = 0;  
            DWORD dwRevision = 0;  
            if (!GetSecurityDescriptorControl(pFileSD, &oldControlBits,  
                &dwRevision)) {  
                    \_tprintf(TEXT("GetSecurityDescriptorControl() failed.")  
                        TEXT("Error %d\n"), GetLastError());  
                    \_\_leave;  
            }  
            if (oldControlBits & SE\_DACL\_AUTO\_INHERITED) {  
                controlBitsOfInterest =  
                    SE\_DACL\_AUTO\_INHERIT\_REQ |  
                    SE\_DACL\_AUTO\_INHERITED ;  
                controlBitsToSet = controlBitsOfInterest;  
            }  
            else if (oldControlBits & SE\_DACL\_PROTECTED) {  
                controlBitsOfInterest = SE\_DACL\_PROTECTED;  
                controlBitsToSet = controlBitsOfInterest;  
            }          
            if (controlBitsOfInterest) {  
                if (!\_SetSecurityDescriptorControl(&newSD,  
                    controlBitsOfInterest,  
                    controlBitsToSet)) {  
                        \_tprintf(TEXT("SetSecurityDescriptorControl() failed.")  
                            TEXT("Error %d\n"), GetLastError());  
                        \_\_leave;  
                }  
            }  
        }  
        //   
        // STEP 20: 把新的SD设置设置到文件的安全属性中（千山万水啊，终于到了）  
        //   
        if (!SetFileSecurity(lpszFileName, secInfo,  
            &newSD)) {  
                \_tprintf(TEXT("SetFileSecurity() failed. Error %d\n"),   
                    GetLastError());  
                \_\_leave;  
        }  
        fResult = TRUE;  
    } \_\_finally {  
        //   
        // STEP 21: 释放已分配的内存，以免Memory Leak  
        //   
        if (pUserSID)  myheapfree(pUserSID);  
        if (szDomain)  myheapfree(szDomain);  
        if (pFileSD) myheapfree(pFileSD);  
        if (pNewACL) myheapfree(pNewACL);  
    }  
    return fResult;  
}  
  
//【接口】  
//     BOOL GetAccountRights(TCHAR \*lpszFileName, TCHAR \*lpszAccountName, int (&arrRights)[32])  
//【机能概要】  
//     获取该文件(目录)指定帐户（组）的权限  
//【入力】  
//     TCHAR \*lpszFileName     文件（目录）  
//     TCHAR \*lpszAccountName  帐户（组）  
//       int (&arrRights)[32]       数组引用，要求传入参数必须是32个int数组  
//【输出】  
//     无  
//【输入输出】  
//     无  
//【返回值】  
//     BOOL  
//【例外】  
//     无  
//---------------------------------------------------------------------------  
BOOL GetAccountRights(TCHAR \*lpszFileName, TCHAR \*lpszAccountName, int (&arrRights)[32]) {  
    //将参数arrRights初始化为0  
    for (int i = 0; i < 32; i++)  
    {  
        arrRights[i] = 0;  
    }  
    // 声明SID变量  
    SID\_NAME\_USE   snuType;  
    // 声明和LookupAccountName相关的变量（注意，全为0，要在程序中动态分配）  
    TCHAR \*        szDomain       = NULL;  
    DWORD          cbDomain       = 0;  
    LPVOID         pUserSID       = NULL;  
    DWORD          cbUserSID      = 0;  
    // 和文件相关的安全描述符 SD 的变量  
    PSECURITY\_DESCRIPTOR pFileSD  = NULL;     // 结构变量  
    DWORD          cbFileSD       = 0;        // SD的size  
    // 和ACL 相关的变量  
    PACL           pACL           = NULL;  
    BOOL           fDaclPresent;  
    BOOL           fDaclDefaulted;  
    ACL\_SIZE\_INFORMATION AclInfo;  
    // 一个临时使用的 ACE 变量  
    LPVOID         pTempAce       = NULL;  
    UINT           CurrentAceIndex = 0;  //ACE在ACL中的位置  
    //API函数的返回值，假设所有的函数都返回失败。  
    BOOL           fResult = FALSE;  
    BOOL           fAPISuccess = FALSE;  
    SECURITY\_INFORMATION secInfo = DACL\_SECURITY\_INFORMATION;  
    \_\_try {  
        //   
        // STEP 1: 通过用户名取得SID  
        //     在这一步中LookupAccountName函数被调用了两次，第一次是取出所需要  
        // 的内存的大小，然后，进行内存分配。第二次调用才是取得了用户的帐户信息。  
        // LookupAccountName同样可以取得域用户或是用户组的信息。（请参看MSDN）  
        //  
        fAPISuccess = LookupAccountName(NULL, lpszAccountName,  
            pUserSID, &cbUserSID, szDomain, &cbDomain, &snuType);  
        // 以上调用API会失败，失败原因是内存不足。并把所需要的内存大小传出。  
        // 下面是处理非内存不足的错误。  
        if (fAPISuccess)  
            \_\_leave;  
        else if (GetLastError() != ERROR\_INSUFFICIENT\_BUFFER) {  
            \_tprintf(TEXT("LookupAccountName() failed. Error %d\n"),   
                GetLastError());  
            \_\_leave;  
        }  
        pUserSID = myheapalloc(cbUserSID);  
        if (!pUserSID) {  
            \_tprintf(TEXT("HeapAlloc() failed. Error %d\n"), GetLastError());  
            \_\_leave;  
        }  
        szDomain = (TCHAR \*) myheapalloc(cbDomain \* sizeof(TCHAR));  
        if (!szDomain) {  
            \_tprintf(TEXT("HeapAlloc() failed. Error %d\n"), GetLastError());  
            \_\_leave;  
        }  
        fAPISuccess = LookupAccountName(NULL, lpszAccountName,  
            pUserSID, &cbUserSID, szDomain, &cbDomain, &snuType);  
        if (!fAPISuccess) {  
            \_tprintf(TEXT("LookupAccountName() failed. Error %d\n"),   
                GetLastError());  
            \_\_leave;  
        }  
        //   
        // STEP 2: 取得文件（目录）相关的安全描述符SD  
        //     使用GetFileSecurity函数取得一份文件SD的拷贝，同样，这个函数也  
        // 是被调用两次，第一次同样是取SD的内存长度。注意，SD有两种格式：自相关的  
        // （self-relative）和 完全的（absolute），GetFileSecurity只能取到“自  
        // 相关的”，而SetFileSecurity则需要完全的。这就是为什么需要一个新的SD，  
        // 而不是直接在GetFileSecurity返回的SD上进行修改。因为“自相关的”信息  
        // 是不完整的。  
        fAPISuccess = GetFileSecurity(lpszFileName,   
            secInfo, pFileSD, 0, &cbFileSD);  
        // 以上调用API会失败，失败原因是内存不足。并把所需要的内存大小传出。  
        // 下面是处理非内存不足的错误。  
        if (fAPISuccess)  
            \_\_leave;  
        else if (GetLastError() != ERROR\_INSUFFICIENT\_BUFFER) {  
            \_tprintf(TEXT("GetFileSecurity() failed. Error %d\n"),   
                GetLastError());  
            \_\_leave;  
        }  
        pFileSD = myheapalloc(cbFileSD);  
        if (!pFileSD) {  
            \_tprintf(TEXT("HeapAlloc() failed. Error %d\n"), GetLastError());  
            \_\_leave;  
        }  
        fAPISuccess = GetFileSecurity(lpszFileName,   
            secInfo, pFileSD, cbFileSD, &cbFileSD);  
        if (!fAPISuccess) {  
            \_tprintf(TEXT("GetFileSecurity() failed. Error %d\n"),   
                GetLastError());  
            \_\_leave;  
        }  
        //   
        // STEP 3: 从GetFileSecurity 返回的SD中取DACL  
        //   
        if (!GetSecurityDescriptorDacl(pFileSD, &fDaclPresent, &pACL,  
            &fDaclDefaulted)) {  
                \_tprintf(TEXT("GetSecurityDescriptorDacl() failed. Error %d\n"),  
                    GetLastError());  
                \_\_leave;  
        }  
        //   
        // STEP 4: 取 DACL的内存size  
        //     GetAclInformation可以提供DACL的内存大小。只传入一个类型为  
        // ACL\_SIZE\_INFORMATION的structure的参数，需DACL的信息，是为了  
        // 方便我们遍历其中的ACE。  
        AclInfo.AceCount = 0; // Assume NULL DACL.  
        AclInfo.AclBytesFree = 0;  
        AclInfo.AclBytesInUse = sizeof(ACL);  
        if (pACL == NULL)  
            fDaclPresent = FALSE;  
        // 如果DACL不为空，则取其信息。（大多数情况下“自关联”的DACL为空）  
        if (fDaclPresent) {              
            if (!GetAclInformation(pACL, &AclInfo,   
                sizeof(ACL\_SIZE\_INFORMATION), AclSizeInformation)) {  
                    \_tprintf(TEXT("GetAclInformation() failed. Error %d\n"),  
                        GetLastError());  
                    \_\_leave;  
            }  
        }  
        //   
        // STEP 5  如果文件（目录） DACL 有数据，将指定帐户的ACE的访问权限转换到整型数组  
        //   
        //     下面的代码假设首先检查指定文件（目录）是否存在的DACL，如果有的话，  
        // 那么就将指定帐户的ACE的访问权限转换到整型数组，我们可以看到其遍历的方法  
        // 是采用ACL\_SIZE\_INFORMATION结构中的AceCount成员来完成的。在这个循环中，  
        // 查找和指定账户相关的ACE  
        //   
        if (fDaclPresent && AclInfo.AceCount) {  
            for (CurrentAceIndex = 0;   
                CurrentAceIndex < AclInfo.AceCount;  
                CurrentAceIndex++) {  
                    //   
                    // STEP 10: 从DACL中取ACE  
                    //   
                    if (!GetAce(pACL, CurrentAceIndex, &pTempAce)) {  
                        \_tprintf(TEXT("GetAce() failed. Error %d\n"),   
                            GetLastError());  
                        \_\_leave;  
                    }  
                    //   
                    //   
                    // STEP 6: 检查要拷贝的ACE的SID是否和需要加入的ACE的SID一样，  
                    // 如果一样，那么就将该ACE的访问权限转换到整型数组，  
                    // 否则跳过，进行下一个循环  
                    //   
                    int nAceType = 1;  
                    if (EqualSid(pUserSID,  
                        &(((ACCESS\_ALLOWED\_ACE \*)pTempAce)->SidStart)))  
                    {  
                        if(((PACE\_HEADER)pTempAce)->AceType == ACCESS\_DENIED\_ACE\_TYPE)  
                        {  
                            nAceType = 2;  
                        }  
                        else  
                        {  
                            nAceType = 1;  
                        }  
                        //bitset类代表的整型数值的顺序是从0到N-1  
                        bitset<32> bitAccessMask(((ACCESS\_ALLOWED\_ACE\*)pTempAce)->Mask);  
                        for (int i = 0; i < 32; i++)  
                        {  
                            if (bitAccessMask[i] != 0 && arrRights[i] != 2)  
                            {  
                                arrRights[i] = nAceType;  
                            }  
                        }  
                    }  
                    else  
                    {  
                        continue;  
                    }  
            }  
        }  
        fResult = TRUE;  
    } \_\_finally {  
        //   
        // STEP 7: 释放已分配的内存，以免Memory Leak  
        //   
        if (pUserSID)  myheapfree(pUserSID);  
        if (szDomain)  myheapfree(szDomain);  
        if (pFileSD) myheapfree(pFileSD);  
    }  
    return fResult;  
}  
  
int \_tmain(int argc, TCHAR \*argv[])   
{  
    if (argc < 3) {  
        \_tprintf(TEXT("usage: \"%s\" <FileName> <AccountName>\n"), argv[0]);  
        return 1;  
    }  
      
    //关于ACCESS\_MASK中各个位代表的含义请参考MSDN  
    string filedesc[] = {"FILE\_READ\_DATA", "FILE\_WRITE\_DATA", "FILE\_APPEND\_DATA", "FILE\_READ\_EA",  
  
        "FILE\_WRITE\_EA", "FILE\_EXECUTE", "FILE\_DELETE\_CHILD", "FILE\_READ\_ATTRIBUTES",  
  
        "FILE\_WRITE\_ATTRIBUTES", " ", " ", " ",  
  
        " ", " ", " ", " ",  
  
        "DELETE ", "READ\_CONTROL", "WRITE\_DAC", "WRITE\_OWNER",  
  
        "SYNCHRONIZE ", " ", " "," ",  
  
        "ACCESS\_SYSTEM\_SECURITY", "MAXIMUM\_ALLOWED", " "," ",  
  
        "GENERIC\_ALL", "GENERIC\_EXECUTE", "GENERIC\_WRITE","GENERIC\_READ"};  
  
    string rights[] = {"Allow", "Deny"};  
  
    //获取ACE中的访问权限  
    //ACE中的访问权限是通过DWORD类型的ACCESS\_MASK记录的  
    //GetAccountRights函数将ACCESS\_MASK转为了一个32个元素的整型数组，并传出  
    int arrRights[32] = {0};  
    if (!GetAccountRights(argv[1], argv[2], arrRights))  
    {  
        \_tprintf(TEXT("GetAccountRights() failed.\n"));  
    }  
    else  
    {  
        \_tprintf(TEXT("The access rights of the file is..\n"));  
        for (int i = 0; i < 32; i++)  
        {  
            int nTmpRight = arrRights[i];  
            if (nTmpRight != 0)  
            {  
                cout<<filedesc[i]<<":  "<<rights[nTmpRight-1]<<endl;  
            }  
        }  
    }  
  
    // argv[1] – 文件（目录）名  
    // argv[2] – 用户（组）名  
    // GENERIC\_ALL表示所有的权限，其是一系列的NTFS权限的或  
    //      NTFS的文件权限很细，还请参看MSDN。  
    if (!AddAccessRights(argv[1], argv[2], GENERIC\_ALL)) {  
        \_tprintf(TEXT("AddAccessRights() failed.\n"));  
        return 1;  
    }  
    else {  
        \_tprintf(TEXT("AddAccessRights() succeeded.\n"));  
        return 0;  
    }  
}

函数AddAccessRights实现为文件(目录)添加一个帐户（组）的权限  
函数GetAccountRights实现获取该文件(目录)指定帐户（组）的权限

**三、**             **一些相关的API函数**

通过以上的示例，相信你已知道如何操作NTFS文件安全属性了，还有一些API函数需要介绍一下。

1、  如果你要加入一个Access-Denied 的ACE，你可以使用**AddAccessDeniedAce**函数

2、  如果你要删除一个ACE，你可以使用**DeleteAce**函数

3、  如果你要检查你所设置的ACL是否合法，你可以使用**IsValidAcl**函数，同样，对于SD的合法也有一个叫**IsValidSecurityDescriptor**的函数

4、  **MakeAbsoluteSD**和**MakeSelfRelativeSD**两个函数可以在两种SD的格式中进行转换。

5、  使用**SetSecurityDescriptorDacl** 和 **SetSecurityDescriptorSacl**可以方便地把ACL设置到SD中。

6、  使用**GetSecurityDescriptorDacl** or **GetSecurityDescriptorSacl**可以方便地取得SD中的ACL结构。

我们把一干和SD/ACL/ACE相关的API函数叫作Low-Level Security Descriptor Functions，其详细信息还请参看MSDN。