www.linuxidc.com

以程序的方式操纵 NTFS 的文件权限

作者: 陈皓

Windows NT/2K/XP版本的操作系统都支持NTFS格式的文件系统，这是一个有安全性质的文件系统，你可以通过Windows的资源管理器来设置对每个目录和文件的用户访问权限。这里我就不对NTFS的安全性进行讲述了，我默认你对NTFS的文件目录的安全设置有了一定的了解。在这里，我将向你介绍使用Windows的API函数来操纵NTFS的文件权限。 www.linuxidc.com

一、理论和术语

在 Windows NT/2K?XP 下的对象，不一定是文件系统，还有其它的一些对象，如：进程、命名管道、打印机、网络共享、或是注册表等等，都可以设置用户访问权限。在 Windows 系统中，其是用一个安全描述符（Security Descriptors）的结构来保存其权限的设置信息，简称为 SD，其在 Windows SDK 中的结构名是“SECURITY_DESCRIPTOR”，这是包括了安全设置信息的结构体。一个安全描述符包含以下信息：

- 一个安全标识符(Security identifiers)，其标识了该信息是哪个对象的，也就是用于记录安全对象的 ID。简称为：SID。
- 一个 DACL（Discretionary Access Control List），其指出了允许和拒绝某用户或用户组的存取控制列表。 当一个进程需要访问安全对象，系统就会检查 DACL 来决定进程的访问权。如果一个对象没有 DACL，那么就是说这个对象是任何人都可以拥有完全的访问权限。

Linux公社(LinuxIDC.com) 是包括Ubuntu,Fedora,SUSE技术，最新IT资讯等Linux专业类网站。

www.linuxidc.com

- 一个 SACL（System Access Control List），其指出了在该对象上的一组存取方式（如，读、写、运行等）的存取控制权限细节的列表。
- 还有其自身的一些控制位。

DACL 和 SACL 构成了整个存取控制列表 Access Control List，简称 ACL，ACL 中的每一项，我们叫做 ACE（Access Control Entry），ACL 中的每一个 ACE。

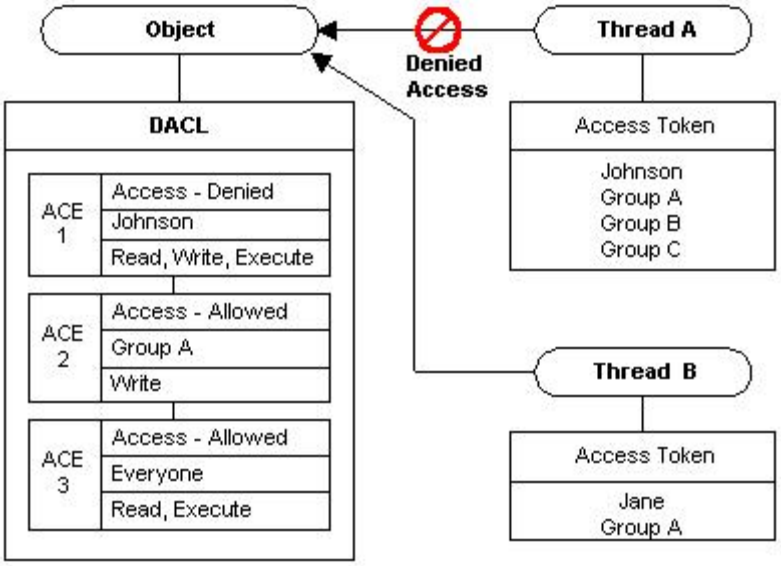
我们的程序不用直接维护 SD 这个结构，这个结构由系统维护。我们只用使用 Windows 提供的相关的 API 函数来取得并设置 SD 中的信息就行了。不过这些 API 函数只有 Windows NT/2K/XP 才支持。

安全对象 Securable Object 是拥有 SD 的 Windows 的对象。所有的被命名的 Windows 的对象都是安全对象。一些没有命名的对象是安全对象，如：进程和线程，也有安全描述符 SD。在对大多数的创建安全对象的操作中都需要你传递一个 SD 的参数，如：CreateFile 和 CreateProcess 函数。另外，Windows 还提供了一系列有关安全对象的安全信息的存取函数，以供你取得对象上的安全设置，或修改对象上的安全设置。如：GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo。

下图说明了，安全对象和 DACL 以及访问者之间的联系（来源于 MSDN）。注意，DACL 表中的每个 ACE 的顺序是有意义的，如果前面的 Allow（或 denied）ACE 通过了，那么，系统就不会检查后面的 ACE 了。

Linux公社(LinuxIDC.com) 是包括Ubuntu,Fedora,SUSE技术，最新IT资讯等Linux专业类网站。

www.linuxidc.com



系统会按照顺序依次检查所有的 ACE 规则，如下面的条件满足，则退出：

1、 如果一个 Access-Denied 的 ACE 明显地拒绝了请求者。

2、 如果某 Access-Allowed 的 ACE 明显地同意了请求者。

3、 全部的ACE都检查完了，但是没有一条ACE明显地允许或是拒绝请求者，那么系统将使用默认值，拒绝请求者的访问。www.linuxidc.com

更多的理论和描述，请参看 MSDN。

二、实践与例程

1、 例程一：创建一个有权限设置的目录

```
#include <windows.h>

void main(void)
{
    SECURITY_ATTRIBUTES sa;   //和文件有关的安全结构
    SECURITY_DESCRIPTOR sd;   //声明一个 SD
```

Linux公社(LinuxIDC.com) 是包括Ubuntu,Fedora,SUSE技术，最新IT资讯等Linux专业类网站。

www.linuxidc.com


```
    BYTE aclBuffer[1024];
    PACL pacl=(PACL)&aclBuffer;  //声明一个 ACL，长度是 1024

    BYTE sidBuffer[100];
    PSID psid=(PSID) &sidBuffer;   //声明一个 SID，长度是 100

    DWORD sidBufferSize = 100;
    char domainBuffer[80];
    DWORD domainBufferSize = 80;
    SID_NAME_USE snu;
    HANDLE file;

    //初始化一个 SD
    InitializeSecurityDescriptor(&sd, SECURITY_DESCRIPTOR_REVISION);
    //初始化一个 ACL
    InitializeAcl(pacl, 1024, ACL_REVISION);
    //查找一个用户 hchen，并取该用户的 SID
    LookupAccountName(0, "hchen", psid,
            &sidBufferSize, domainBuffer,
            &domainBufferSize, &snu);
    //设置该用户的 Access-Allowed 的 ACE，其权限为“所有权限”
    AddAccessAllowedAce(pacl, ACL_REVISION, GENERIC_ALL, psid);
    //把 ACL 设置到 SD 中
    SetSecurityDescriptorDacl(&sd, TRUE, pacl, FALSE);

    //把 SD 放到文件安全结构 SA 中
    sa.nLength = sizeof(SECURITY_ATTRIBUTES);
    sa.bInheritHandle = FALSE;
    sa.lpSecurityDescriptor = &sd;

    //创建文件
    file = CreateFile("c:\\testfile",
        0, 0, &sa, CREATE_NEW, FILE_ATTRIBUTE_NORMAL, 0);
    CloseHandle(file);
}
```

这个例子我是从网上找来的，改了改。其中使用到的关键的 API 函数，我都把其加粗了。从程序中我们可以看到，我们先初始化了一个 SD 和一个 ACL，然后调用 LookupAccountName 取得用户的 SID，然后通过这个 SID，对 ACL 中加入一个有允许访问权限的 ACE，然后再把整个 ACL 设置到 SD 中。最后，组织文件安全

Linux公社(LinuxIDC.com) 是包括Ubuntu,Fedora,SUSE技术，最新IT资讯等Linux专业类网站。

www.linuxidc.com

描述的 SA 结构，并调用 CreateFile 创建文件。如果你的操作系统是 NTFS，那么，你可以看到你创建出来的文件的安全属性的样子：



这个程序旨在说明如何生成一个新的 SD 和 ACL 的用法，其有四个地方的不足和不清：

1、 对于 ACL 和 SID 的声明采用了硬编码的方式指定其长度。

2、 对于 API 函数，没有出错处理。

3、 没有说明如何修改已有文件或目录的安全设置。

4、 没有说明安全设置的继承性。

对于这些我将在下个例程中讲述。

2、 例程二、为目录增加一个安全设置项

Linux公社(LinuxIDC.com) 是包括Ubuntu,Fedora,SUSE技术，最新IT资讯等Linux专业类网站。

www.linuxidc.com

在我把这个例程序例出来以前，请允许我多说一下。

1、 对于文件、目录、命令管道，我们不一定要使用GetNamedSecurityInfo和SetNamedSecurityInfo函数，我们可以使用其专用函数GetFileSecurity和SetFileSecurity函数来取得或设置文件对象的SD，以设置其访问权限。需要使用这两个函数并不容易，正如前面我们所说的，我们还需要处理SD参数，要处理SD，就需要处理DACL和ACE，以及用户的相关SID，于是，一系统列的函数就被这两个函数带出来了。www.linuxidc.com

2、 对于上一个例子中的使用硬编码指定 SID 的处理方法是。调用 LookupAccountName 函数时，先把 SID，Domain 名的参数传为空 NULL，于是 LookupAccountName 会返回用户的 SID 的长度和 Domain 名的长度，于是你可以根据这个长度分配内存，然后再次调用 LookupAccountName 函数。于是就可以达到到态分配内存的效果。对于 ACL 也一样。

3、 对于给文件的 ACL 中增加一个 ACE 条目，一般的做法是先取出文件上的 ACL，逐条取出 ACE，和现需要增加的 ACE 比较，如果有冲突，则删除已有的 ACE，把新加的 ACE 添置到最后。这里的最后，应该是非继承而来的 ACE 的最后。关于 ACL 继承，NTFS 中，你可以设置文件和目录是否继承于其父目录的设置。在程序中同样可以设置。

还是请看例程，这个程序比较长，来源于 MSDN，我做了一点点修改，并把自己的理解加在注释中，所以，请注意代码中的注释：

```
#include <windows.h>
#include <tchar.h>
#include <stdio.h>
```

Linux公社(LinuxIDC.com) 是包括Ubuntu,Fedora,SUSE技术，最新IT资讯等Linux专业类网站。

www.linuxidc.com


```
//使用 Windows 的 HeapAlloc 函数进行动态内存分配
#define myheapalloc(x) (HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, x))
#define myheapfree(x)  (HeapFree(GetProcessHeap(), 0, x))
typedef BOOL (WINAPI *SetSecurityDescriptorControlFnPtr)(
   IN PSECURITY_DESCRIPTOR pSecurityDescriptor,
   IN SECURITY_DESCRIPTOR_CONTROL ControlBitsOfInterest,
   IN SECURITY_DESCRIPTOR_CONTROL ControlBitsToSet);
typedef BOOL (WINAPI *AddAccessAllowedAceExFnPtr)(
  PACL pAcl,
  DWORD dwAceRevision,
  DWORD AceFlags,
  DWORD AccessMask,
  PSID pSid
);
BOOL AddAccessRights(TCHAR *lpszFileName, TCHAR *lpszAccountName,
      DWORD dwAccessMask) {
   // 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
   // 一个新的 SD 的变量，用于构造新的 ACL（把已有的 ACL 和需要新加的 ACL 整合起来）
   SECURITY_DESCRIPTOR  newSD;
   // 和 ACL 相关的变量
   PACL           pACL           = NULL;
   BOOL           fDaclPresent;
   BOOL           fDaclDefaulted;
   ACL_SIZE_INFORMATION AclInfo;
   // 一个新的 ACL 变量
   PACL           pNewACL        = NULL;  //结构指针变量
   DWORD          cbNewACL       = 0;     //ACL 的 size
   // 一个临时使用的 ACE 变量
   LPVOID         pTempAce       = NULL;
   UINT           CurrentAceIndex = 0;  //ACE 在 ACL 中的位置
   UINT           newAceIndex = 0;  //新添的 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;
   BOOL           fAPISuccess;
   SECURITY_INFORMATION secInfo = DACL_SECURITY_INFORMATION;
```

Linux公社(LinuxIDC.com) 是包括Ubuntu,Fedora,SUSE技术，最新IT资讯等Linux专业类网站。

www.linuxidc.com


```
// 下面的两个函数是新的 API 函数，仅在 Windows 2000 以上版本的操作系统支持。
// 在此将从 Advapi32.dll 文件中动态载入。如果你使用 VC++ 6.0 编译程序，而且你想
// 使用这两个函数的静态链接。则请为你的编译加上：/D_WIN32_WINNT=0x0500
// 的编译参数。并且确保你的 SDK 的头文件和 lib 文件是最新的。
SetSecurityDescriptorControlFnPtr _SetSecurityDescriptorControl = NULL;
AddAccessAllowedAceExFnPtr _AddAccessAllowedAceEx = NULL;
__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
```

Linux公社(LinuxIDC.com) 是包括Ubuntu,Fedora,SUSE技术，最新IT资讯等Linux专业类网站。

www.linuxidc.com


```
    // 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: 初始化一个新的 SD
    //
    if (!InitializeSecurityDescriptor(&newSD,
          SECURITY_DESCRIPTOR_REVISION)) {
       _tprintf(TEXT("InitializeSecurityDescriptor() failed.")
          TEXT("Error %d\n"), GetLastError());
       __leave;
    }
    //
    // STEP 4: 从 GetFileSecurity 返回的 SD 中取 DACL
    //
    if (!GetSecurityDescriptorDacl(pFileSD, &fDaclPresent, &pACL,
          &fDaclDefaulted)) {
       _tprintf(TEXT("GetSecurityDescriptorDacl() failed. Error %d\n"),
             GetLastError());
```

Linux公社(LinuxIDC.com) 是包括Ubuntu,Fedora,SUSE技术，最新IT资讯等Linux专业类网站。

www.linuxidc.com


```
    __leave;
}
//
// STEP 5: 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 6: 计算新的 ACL 的 size
//    计算的公式是：原有的 DACL 的 size 加上需要添加的一个 ACE 的 size，以
// 及加上一个和 ACE 相关的 SID 的 size，最后减去两个字节以获得精确的大小。
cbNewACL = AclInfo.AclBytesInUse + sizeof(ACCESS_ALLOWED_ACE)
      + GetLengthSid(pUserSID) - sizeof(DWORD);

//
// STEP 7: 为新的 ACL 分配内存
//
pNewACL = (PACL) myheapalloc(cbNewACL);
if (!pNewACL) {
   _tprintf(TEXT("HeapAlloc() failed. Error %d\n"), GetLastError());
   __leave;
}
//
// STEP 8: 初始化新的 ACL 结构
//
if (!InitializeAcl(pNewACL, cbNewACL, ACL_REVISION2)) {
   _tprintf(TEXT("InitializeAcl() failed. Error %d\n"),
         GetLastError());
   __leave;
}
```

Linux公社(LinuxIDC.com) 是包括Ubuntu,Fedora,SUSE技术，最新IT资讯等Linux专业类网站。

www.linuxidc.com

(程序未完，请看下篇)

```
    //
    // STEP 9  如果文件（目录） DACL 有数据，拷贝其中的 ACE 到新的 DACL 中
    //
    //     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，
    // 那么就拷贝所有的 ACE 到新的 DACL 结构中，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，
    // 会按照默认的 ACE 的顺序来进行拷贝（ACE 在 ACL 中的顺序是很关键的），在拷
    // 贝过程中，先拷贝非继承的 ACE（我们知道 ACE 会从上层目录中继承下来）
    //
    newAceIndex = 0;
    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 11: 检查是否是非继承的 ACE
          //     如果当前的 ACE 是一个从父目录继承来的 ACE，那么就退出循环。
          // 因为，继承的 ACE 总是在非继承的 ACE 之后，而我们所要添加的 ACE
          // 应该在已有的非继承的 ACE 之后，所有的继承的 ACE 之前。退出循环
          // 正是为了要添加一个新的 ACE 到新的 DACL 中，这后，我们再把继承的
          // ACE 拷贝到新的 DACL 中。
          //
          if (((ACCESS_ALLOWED_ACE *)pTempAce)->Header.AceFlags
             & INHERITED_ACE)
             break;
          //
          // STEP 12: 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，那么就应该废掉已存在的 ACE，也就是说，同一个用户的存取
          // 权限的设置的 ACE，在 DACL 中应该唯一。这在里，跳过对同一用户已设置
          // 了的 ACE，仅是拷贝其它用户的 ACE。
          //
          if (EqualSid(pUserSID,
             &(((ACCESS_ALLOWED_ACE *)pTempAce)->SidStart)))
             continue;
          //
```

Linux公社(LinuxIDC.com) 是包括Ubuntu,Fedora,SUSE技术，最新IT资讯等Linux专业类网站。

www.linuxidc.com


```
            // STEP 13: 把 ACE 加入到新的 DACL 中
            //    下面的代码中，注意 AddAce 函数的第三个参数，这个参数的意思是
            // ACL 中的索引值，意为要把 ACE 加到某索引位置之后，参数 MAXDWORD 的
              // 意思是确保当前的 ACE 是被加入到最后的位置。
            //
            if (!AddAce(pNewACL, ACL_REVISION, MAXDWORD, pTempAce,
                  ((PACE_HEADER) pTempAce)->AceSize)) {
               _tprintf(TEXT("AddAce() failed. Error %d\n"),
                     GetLastError());
               __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());
```

Linux公社(LinuxIDC.com) 是包括Ubuntu,Fedora,SUSE技术，最新IT资讯等Linux专业类网站。

www.linuxidc.com

```
            __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
    //
    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。
    //
```

www.linuxidc.com


```
_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;
```

Linux公社(LinuxIDC.com) 是包括Ubuntu,Fedora,SUSE技术，最新IT资讯等Linux专业类网站。

www.linuxidc.com


```
    } __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;
}

int _tmain(int argc, TCHAR *argv[]) {
    if (argc < 3) {
        _tprintf(TEXT("usage: \"%s\" <FileName> <AccountName>\n"), argv[0]);
        return 1;
    }
    // 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;
    }
}
```

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。

Linux公社(LinuxIDC.com) 是包括Ubuntu,Fedora,SUSE技术，最新IT资讯等Linux专业类网站。

www.linuxidc.com

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。



Linux公社(LinuxIDC.com) 是包括Ubuntu,Fedora,SUSE技术，最新IT资讯等Linux专业类网站。