Another way to create a process with attributes, maybe worse maybe better

devblogs.microsoft.com/oldnewthing/20130426-00

April 26, 2013

Raymond Chen

Adam Rosenfield noted that "those sure are a lot of hoops you have to jump through to solve this unusual problem" of specifying which handles are inherited by a new process.

Well, first of all, what's so wrong with that? You have to jump through a lot of hoops when you are in an unusual situation. But by definition, most people are not in an unusual situation, so it's an instance of the *Pay for Play* principle: The simple case should be easy, and it's okay for the complicated case to be hard. (It's usually difficult to make the complicated case easy; that's why it's called the complicated case.)

The complexity mostly comes from managing the general-purpose `PROC_THREAD_ATTRIBUTE_LIST`, which is used for things other than just controlling inherited handles. It's a generic way of passing up to *N* additional parameters to `Create-Process` without having to create 2^N different variations of `CreateProcess`.

The `CreateProcessWithExplicitHandles` function was just one of the *N* special-purpose functions that the `PROC_THREAD_ATTRIBUTE_LIST` tried to avoid having to create. And the special-purpose function naturally takes the special-purpose case and applies the general solution to it. It's complicated because you are now doing something complicated.

That said, here's one attempt to make it less complicated: By putting all the complicated stuff closer to the complicated function:

```
typedef struct PROCTHREADATTRIBUTE {
 DWORD_PTR Attribute;
 PVOID lpValue;
 SIZE_T cbSize;
} PROCTHREADATTRIBUTE;


BOOL CreateProcessWithAttributes(
  __in_opt     LPCTSTR lpApplicationName,
  __inout_opt  LPTSTR lpCommandLine,
  __in_opt     LPSECURITY_ATTRIBUTES lpProcessAttributes,
  __in_opt     LPSECURITY_ATTRIBUTES lpThreadAttributes,
  __in         BOOL bInheritHandles,
  __in         DWORD dwCreationFlags,
  __in_opt     LPVOID lpEnvironment,
  __in_opt     LPCTSTR lpCurrentDirectory,
  __in         LPSTARTUPINFO lpStartupInfo,
  __out        LPPROCESS_INFORMATION lpProcessInformation,
    // here is the new stuff
    __in       DWORD cAttributes,
    __in_ecount(cAttributes) const PROCTHREADATTRIBUTE rgAttributes[])
{
 BOOL fSuccess;
 BOOL fInitialized = FALSE;
 SIZE_T size = 0;
 LPPROC_THREAD_ATTRIBUTE_LIST lpAttributeList = NULL;


 fSuccess = InitializeProcThreadAttributeList(NULL, cAttributes, 0, &size) ||
            GetLastError() == ERROR_INSUFFICIENT_BUFFER;


 if (fSuccess) {
  lpAttributeList = reinterpret_cast<LPPROC_THREAD_ATTRIBUTE_LIST>
                              (HeapAlloc(GetProcessHeap(), 0, size));
  fSuccess = lpAttributeList != NULL;
 }
 if (fSuccess) {
  fSuccess = InitializeProcThreadAttributeList(lpAttributeList,
                  cAttributes, 0, &size);
 }
 if (fSuccess) {
  fInitialized = TRUE;
  for (DWORD index = 0; index < cAttributes && fSuccess; index++) {
   fSuccess = UpdateProcThreadAttribute(lpAttributeList,
                     0, rgAttributes[index].Attribute,
                     rgAttributes[index].lpValue,
                     rgAttributes[index].cbSize, NULL, NULL);
  }
 }
 if (fSuccess) {
  STARTUPINFOEX info;
  ZeroMemory(&info, sizeof(info));
  info.StartupInfo = *lpStartupInfo;
  info.StartupInfo.cb = sizeof(info);
  info.lpAttributeList = lpAttributeList;
  fSuccess = CreateProcess(lpApplicationName,
                           lpCommandLine,
                           lpProcessAttributes,
                           lpThreadAttributes,
                           bInheritHandles,
                           dwCreationFlags | EXTENDED_STARTUPINFO_PRESENT,
                           lpEnvironment,
                           lpCurrentDirectory,
                           &info.StartupInfo,
                           lpProcessInformation);
 }


 if (fInitialized) DeleteProcThreadAttributeList(lpAttributeList);
 if (lpAttributeList) HeapFree(GetProcessHeap(), 0, lpAttributeList);
 return fSuccess;
}
```

There, now the complexity is there because you're a generic complex function, so you have no reason to complain.

A caller of this function might go like this:

```
  HANDLE handles[2] = { handle1, handle2 };
  const PROCTHREADATTRIBUTE attributes[] = {
   {
    PROC_THREAD_ATTRIBUTE_HANDLE_LIST,
    handles,
    sizeof(handles),
   },
  };


  fSuccess = CreateProcessWithAttributes(
                           lpApplicationName,
                           lpCommandLine,
                           lpProcessAttributes,
                           lpThreadAttributes,
                           bInheritHandles,
                           dwCreationFlags,
                           lpEnvironment,
                           lpCurrentDirectory,
                           lpStartupInfo,
                           lpProcessInformation,
                           ARRAYSIZE(attributes),
                           attributes);
```

Adam hates the “chained success” style and prefers the “goto” style; on the other hand, other people hate gotos. So to be fair, I will choose a coding style that nobody likes. That way everybody is equally unhappy.

```
BOOL CreateProcessWithAttributes(
  __in_opt     LPCTSTR lpApplicationName,
  __inout_opt  LPTSTR lpCommandLine,
  __in_opt     LPSECURITY_ATTRIBUTES lpProcessAttributes,
  __in_opt     LPSECURITY_ATTRIBUTES lpThreadAttributes,
  __in         BOOL bInheritHandles,
  __in         DWORD dwCreationFlags,
  __in_opt     LPVOID lpEnvironment,
  __in_opt     LPCTSTR lpCurrentDirectory,
  __in         LPSTARTUPINFO lpStartupInfo,
  __out        LPPROCESS_INFORMATION lpProcessInformation,
    // here is the new stuff
    __in       DWORD cAttributes,
    __in_ecount(cAttributes) const PROCTHREADATTRIBUTE rgAttributes[])
{
 BOOL fSuccess = FALSE;
 SIZE_T size = 0;


 if (InitializeProcThreadAttributeList(NULL, cAttributes, 0, &size) ||
     GetLastError() == ERROR_INSUFFICIENT_BUFFER) {
  LPPROC_THREAD_ATTRIBUTE_LIST lpAttributeList =
         reinterpret_cast<LPPROC_THREAD_ATTRIBUTE_LIST>
                               (HeapAlloc(GetProcessHeap(), 0, size));
  if (lpAttributeList != NULL) {
   if (InitializeProcThreadAttributeList(lpAttributeList,
                    cAttributes, 0, &size)) {
    DWORD index;
    for (index = 0;
         index < cAttributes &&
         UpdateProcThreadAttribute(lpAttributeList,
                       0, rgAttributes[index].Attribute,
                       rgAttributes[index].lpValue,
                       rgAttributes[index].cbSize, NULL, NULL);
         index++) {
    }
    if (index >= cAttributes) {
     STARTUPINFOEX info;
     ZeroMemory(&info, sizeof(info));
     info.StartupInfo = *lpStartupInfo;
     info.StartupInfo.cb = sizeof(info);
     info.lpAttributeList = lpAttributeList;
     fSuccess = CreateProcess(
                            lpApplicationName,
                            lpCommandLine,
                            lpProcessAttributes,
                            lpThreadAttributes,
                            bInheritHandles,
                            dwCreationFlags | EXTENDED_STARTUPINFO_PRESENT,
                            lpEnvironment,
                            lpCurrentDirectory,
                            &info.StartupInfo,
                           lpProcessInformation);
    }
    DeleteProcThreadAttributeList(lpAttributeList);
   }
   HeapFree(GetProcessHeap(), 0, lpAttributeList);
  }
 }

 return fSuccess;
}
```

Those who are really adventuresome could try a version of `CreateProcessWith-Attributes` that uses varargs or `std::initializer_list`.

Raymond Chen

Follow