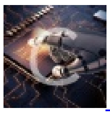
win32程序如何挂起/恢复(suspend/resume)进程

原创

吓人的鸟 于 2012-02-07 16:28:56 发布 4944 收藏 3
分类专栏： 系统-win32 编程语言-c/c++ 代码库/程序片段 文章标签： thread module structure null hex library


系统-win32 同时被 3 个专栏收录
18 篇文章 0 订阅
订阅专栏

编程语言-c/c++
38 篇文章 1 订阅
订阅专栏

代码库/程序片段
24 篇文章 0 订阅
订阅专栏

1.msdn并木有提供现成的类似SuspendProcess/ResumeProcess的API，只能通过SuspendThread/ResumeThread来实现

2.如何显示所有进程？根据进程id如何获取其内部各线程？

这些仅需要熟悉MSDN的Tool Help Library几个函数（**CreateToolhelp32SnapshotProcess32FirstProcess32NextThread32FirstThread32Next**）即可

源文档：http://www.codeproject.com/Articles/2964/Win32-process-suspend-resume-tool

下面是摘录的核心函数

```
BOOL PauseResumeThreadList(DWORD dwOwnerPID, bool bResumeThread)
{
    HANDLE        hThreadSnap = NULL;
    BOOL          bRet        = FALSE;
    THREADENTRY32 te32        = {0};

    // Take a snapshot of all threads currently in the system.

    hThreadSnap = CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD, 0);
    if (hThreadSnap == INVALID_HANDLE_VALUE)
        return (FALSE);

    // Fill in the size of the structure before using it.

    te32.dwSize = sizeof(THREADENTRY32);

    // Walk the thread snapshot to find all threads of the process.
    // If the thread belongs to the process, add its information
    // to the display list.
    if (Thread32First(hThreadSnap, &te32))
    {
        do
        {
            if (te32.th32OwnerProcessID == dwOwnerPID)
            {
    HANDLE hThread = OpenThread(THREAD_SUSPEND_RESUME, FALSE, te32.th32ThreadID);
    if (bResumeThread)
    {
     cout << _T("Resuming Thread 0x") << cout.setf( ios_base::hex ) << te32.th32ThreadID << '\n';
     ResumeThread(hThread);
    }
    else
    {
     cout << _T("Suspending Thread 0x") << cout.setf( ios_base::hex ) << te32.th32ThreadID << '\n';
     SuspendThread(hThread);
    }
    CloseHandle(hThread);
            }
        }
        while (Thread32Next(hThreadSnap, &te32));
        bRet = TRUE;
    }
    else
        bRet = FALSE;          // could not walk the list of threads

    // Do not forget to clean up the snapshot object.
    CloseHandle (hThreadSnap);

    return (bRet);
}

BOOL ProcessList()
{
    HANDLE         hProcessSnap = NULL;
    BOOL           bRet      = FALSE;
    PROCESSENTRY32 pe32      = {0};

    //  Take a snapshot of all processes in the system.
    hProcessSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);

    if (hProcessSnap == INVALID_HANDLE_VALUE)
        return (FALSE);

    //  Fill in the size of the structure before using it.
    pe32.dwSize = sizeof(PROCESSENTRY32);

    //  Walk the snapshot of the processes, and for each process,
    //  display information.

    if (Process32First(hProcessSnap, &pe32))
    {
        do
        {
   cout << _T("PID\t") << pe32.th32ProcessID << '\t' << pe32.szExeFile << '\n';
        }
        while (Process32Next(hProcessSnap, &pe32));
        bRet = TRUE;
    }
    else
        bRet = FALSE;    // could not walk the list of processes

    // Do not forget to clean up the snapshot object.

    CloseHandle (hProcessSnap);
    return (bRet);
}
```

根据程序名称杀进程：

1.取process快照，遍历进程根据进程名称找到要杀的进程疑犯

2.如果根据名称把所有疑犯通杀就可能会杀掉其他跟自己无关但同名的程序，此时可以指定杀死某些路径下启动的进程，如何做到呢？

如果是vista及以上的系统可以用QueryFullProcessImageName找到该进程的路径名，判断是否该杀

如果需要支持更低版本的系统就比较麻烦一点此时需要获取module快照（第二个参数需要指定为嫌疑进程的id，否则当该嫌疑进程非主进程时module快照是找不到它的），然后遍历module，找到进程id等于疑犯进程的module，取其路径，如果符合条件就杀掉疑犯进程

下面是后面这种情况的实现代码

```
int KillProcessByName(const CString& processName, const CString& processPath)
{
 //return 0（进程不存在） 1（进程存在并被杀死） -1（进程存在杀不死）
 int result = 0;

 HANDLE         hProcessSnap = NULL;
 BOOL           bRet      = FALSE;
 PROCESSENTRY32 pe32      = {0};

 hProcessSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);

 if (hProcessSnap == INVALID_HANDLE_VALUE) {
  ASSERT(false);
  return 0;
 }

 pe32.dwSize = sizeof(PROCESSENTRY32);

 if (Process32First(hProcessSnap, &pe32))
 {
  do {
   if (processName.Compare(pe32.szExeFile) == 0) {
    HANDLE tmpProc = OpenProcess(PROCESS_TERMINATE , FALSE, pe32.th32ProcessID);
    if (!tmpProc) {
     ASSERT(false);
     continue;
    }
    if (!processPath.IsEmpty()) {
     HANDLE hModuleSnap = INVALID_HANDLE_VALUE;
     MODULEENTRY32 me32;
     hModuleSnap = CreateToolhelp32Snapshot( TH32CS_SNAPMODULE, pe32.th32ProcessID );
     if( hModuleSnap == INVALID_HANDLE_VALUE )
     {
      ASSERT(false);
      continue;
     }

     me32.dwSize = sizeof( MODULEENTRY32 );

     if( !Module32First( hModuleSnap, &me32 ) )
     {
      ASSERT(false);
      CloseHandle( hModuleSnap );
      continue;
     }

     do {
      if (me32.th32ProcessID == pe32.th32ProcessID) {
       CString realPath = me32.szExePath;
       if (realPath.FindOneOf(processPath) == 0) {
        BOOL rt = TerminateProcess(tmpProc, NULL);
        if (rt == 0)
         result = -1;
        else if (result != -1)
         result = 1;
       }
       break;
      }
     } while( Module32Next( hModuleSnap, &me32 ) );
     CloseHandle( hModuleSnap );
    } else {
     BOOL rt = TerminateProcess(tmpProc, NULL);
     if (rt == 0)
      result = -1;
     else if (result != -1)
      result = 1;
    }
   }
  }
  while (Process32Next(hProcessSnap, &pe32));
 }
 else {
  ASSERT(false);
 }

 CloseHandle (hProcessSnap);
 return result;
}
```