Taking a snapshot, viewing processes - Win32 apps

By Karl-Bridge-Microsoft

Archived: 2026-04-06 15:39:37 UTC

This code example retrieves a list of running processes. First, the **GetProcessList** function takes a snapshot of currently executing processes in the system. To do that, it uses the **CreateToolhelp32Snapshot** function, and then it walks through the list recorded in the snapshot by using **Process32First** and **Process32Next**. For each process in turn, **GetProcessList** calls the **ListProcessModules** function, which is described in Traversing the module list, and the **ListProcessThreads** function, which is described in Traversing the thread list.

A simple error-reporting function, **printError**, displays the reason for any failures (which typically result from security restrictions). For example, **OpenProcess** fails for the Idle and CSRSS processes because their access restrictions prevent user-level code from opening them.

To follow along with the code example, use Visual Studio to create a new project from the C++ **Console App** project template, and add the code below to it.

```
 #include <windows.h>
#include <tlhelp32.h>
#include <tchar.h>
#include <stdio.h>

//  Forward declarations:
BOOL GetProcessList( );
BOOL ListProcessModules( DWORD dwPID );
BOOL ListProcessThreads( DWORD dwOwnerPID );
void printError( TCHAR const* msg );

int main( void )
{
  GetProcessList( );
  return 0;
}

BOOL GetProcessList( )
{
  HANDLE hProcessSnap;
  HANDLE hProcess;
  PROCESSENTRY32 pe32;
  DWORD dwPriorityClass;

  // Take a snapshot of all processes in the system.
```

```
hProcessSnap = CreateToolhelp32Snapshot( TH32CS_SNAPPROCESS, 0 );
if( hProcessSnap == INVALID_HANDLE_VALUE )
{
  printError( TEXT("CreateToolhelp32Snapshot (of processes)") );
  return( FALSE );
}

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

// Retrieve information about the first process,
// and exit if unsuccessful
if( !Process32First( hProcessSnap, &pe32 ) )
{
  printError( TEXT("Process32First") ); // show cause of failure
  CloseHandle( hProcessSnap );          // clean the snapshot object
  return( FALSE );
}

// Now walk the snapshot of processes, and
// display information about each process in turn
do
{
  _tprintf( TEXT("\n\n=====================================================" ));
  _tprintf( TEXT("\nPROCESS NAME:  %s"), pe32.szExeFile );
  _tprintf( TEXT("\n-----------------------------------------------------" ));

  // Retrieve the priority class.
  dwPriorityClass = 0;
  hProcess = OpenProcess( PROCESS_ALL_ACCESS, FALSE, pe32.th32ProcessID );
  if( hProcess == NULL )
    printError( TEXT("OpenProcess") );
  else
  {
    dwPriorityClass = GetPriorityClass( hProcess );
    if( !dwPriorityClass )
      printError( TEXT("GetPriorityClass") );
    CloseHandle( hProcess );
  }

  _tprintf( TEXT("\n  Process ID        = 0x%08X"), pe32.th32ProcessID );
  _tprintf( TEXT("\n  Thread count      = %d"),   pe32.cntThreads );
  _tprintf( TEXT("\n  Parent process ID = 0x%08X"), pe32.th32ParentProcessID );
  _tprintf( TEXT("\n  Priority base     = %d"), pe32.pcPriClassBase );
  if( dwPriorityClass )
    _tprintf( TEXT("\n  Priority class    = %d"), dwPriorityClass );
```

```
    // List the modules and threads associated with this process
    ListProcessModules( pe32.th32ProcessID );
    ListProcessThreads( pe32.th32ProcessID );

  } while( Process32Next( hProcessSnap, &pe32 ) );

  CloseHandle( hProcessSnap );
  return( TRUE );
}


BOOL ListProcessModules( DWORD dwPID )
{
  HANDLE hModuleSnap = INVALID_HANDLE_VALUE;
  MODULEENTRY32 me32;

  // Take a snapshot of all modules in the specified process.
  hModuleSnap = CreateToolhelp32Snapshot( TH32CS_SNAPMODULE, dwPID );
  if( hModuleSnap == INVALID_HANDLE_VALUE )
  {
    printError( TEXT("CreateToolhelp32Snapshot (of modules)") );
    return( FALSE );
  }

  // Set the size of the structure before using it.
  me32.dwSize = sizeof( MODULEENTRY32 );

  // Retrieve information about the first module,
  // and exit if unsuccessful
  if( !Module32First( hModuleSnap, &me32 ) )
  {
    printError( TEXT("Module32First") );  // show cause of failure
    CloseHandle( hModuleSnap );           // clean the snapshot object
    return( FALSE );
  }

  // Now walk the module list of the process,
  // and display information about each module
  do
  {
    _tprintf( TEXT("\n\n     MODULE NAME:     %s"),   me32.szModule );
    _tprintf( TEXT("\n     Executable     = %s"),     me32.szExePath );
    _tprintf( TEXT("\n     Process ID     = 0x%08X"),         me32.th32ProcessID );
    _tprintf( TEXT("\n     Ref count (g)  = 0x%04X"),     me32.GlblcntUsage );
    _tprintf( TEXT("\n     Ref count (p)  = 0x%04X"),     me32.ProccntUsage );
    _tprintf( TEXT("\n     Base address   = 0x%08X"), (DWORD) me32.modBaseAddr );
    _tprintf( TEXT("\n     Base size      = %d"),             me32.modBaseSize );
```

```
  } while( Module32Next( hModuleSnap, &me32 ) );

  CloseHandle( hModuleSnap );
  return( TRUE );
}

BOOL ListProcessThreads( DWORD dwOwnerPID )
{
  HANDLE hThreadSnap = INVALID_HANDLE_VALUE;
  THREADENTRY32 te32;

  // Take a snapshot of all running threads
  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);

  // Retrieve information about the first thread,
  // and exit if unsuccessful
  if( !Thread32First( hThreadSnap, &te32 ) )
  {
    printError( TEXT("Thread32First") ); // show cause of failure
    CloseHandle( hThreadSnap );          // clean the snapshot object
    return( FALSE );
  }

  // Now walk the thread list of the system,
  // and display information about each thread
  // associated with the specified process
  do
  {
    if( te32.th32OwnerProcessID == dwOwnerPID )
    {
      _tprintf( TEXT("\n\n     THREAD ID      = 0x%08X"), te32.th32ThreadID );
      _tprintf( TEXT("\n     Base priority  = %d"), te32.tpBasePri );
      _tprintf( TEXT("\n     Delta priority = %d"), te32.tpDeltaPri );
      _tprintf( TEXT("\n"));
    }
  } while( Thread32Next(hThreadSnap, &te32 ) );

  CloseHandle( hThreadSnap );
  return( TRUE );
}
```

```
void printError( TCHAR const* msg )
{
  DWORD eNum;
  TCHAR sysMsg[256];
  TCHAR* p;

  eNum = GetLastError( );
  FormatMessage( FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS,
         NULL, eNum,
         MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
         sysMsg, 256, NULL );

  // Trim the end of the line and terminate it with a null
  p = sysMsg;
  while( ( *p > 31 ) || ( *p == 9 ) )
    ++p;
  do { *p-- = 0; } while( ( p >= sysMsg ) &&
                          ( ( *p == '.' ) || ( *p < 33 ) ) );

  // Display the message
  _tprintf( TEXT("\n  WARNING: %s failed with error %d (%s)"), msg, eNum, sysMsg );
}
```

- Snapshots of the system

Source: https://msdn.microsoft.com/library/windows/desktop/ms686701.aspx