**Instituto Politécnico Nacional**

**UNIDAD PROFESIONAL INTERDISCIPLINARIA EN**

**INGENIERÍA Y TECNOLOGÍAS AVANZADAS**

**Ingeniería Mecatrónica**

**Programación Avanzada.**

**Función CreateProcess**

**Alumno**:

* Calderón Juárez Rodrigo Alexis

**Grupo:** 2MV4

**Nombre del profesor:** Maza Casas Lamberto

Fecha: 14/04/18

Ciudad de México

**CreateProcess**

La función CreateProcess crea un nuevo proceso, que se ejecuta independientemente del proceso de creación. Sin embargo, para simplificar, la relación se conoce como una relación padre-hijo.

Crea un nuevo proceso y su hilo principal. El nuevo proceso se ejecuta en el contexto de seguridad del proceso de llamada. Si el proceso de llamada se hace pasar por otro usuario, el nuevo proceso usa el token para el proceso de llamada, no el token de suplantación.

**Sintaxis**

C++

BOOL WINAPI CreateProcess(

\_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

);

**Parámetros**

lpApplicationName [en, opcional]

El nombre del módulo a ser ejecutado. Este módulo puede ser una aplicación basada en Windows. Puede ser algún otro tipo de módulo (por ejemplo, MS-DOS u OS / 2) si el subsistema apropiado está disponible en la computadora local.

La cadena puede especificar la ruta completa y el nombre de archivo del módulo a ejecutar o puede especificar un nombre parcial. En el caso de un nombre parcial, la función usa la unidad actual y el directorio actual para completar la especificación. La función no usará la ruta de búsqueda. Este parámetro debe incluir la extensión del nombre de archivo; no se asume ninguna extensión por defecto

El parámetro lpApplicationName puede ser NULL. En ese caso, el nombre del módulo debe ser el primer token blanco delimitado por espacios en la cadena lpCommandLine. Si está utilizando un nombre de archivo largo que contiene un espacio, use cadenas entre comillas para indicar dónde termina el nombre del archivo y comienzan los argumentos; de lo contrario, el nombre del archivo es ambiguo. Por ejemplo, considere la cadena "c: \ archivos de programa \ subdirectorio \ nombre de programa". Esta cadena se puede interpretar de varias maneras. El sistema intenta interpretar las posibilidades en el siguiente orden:

**c:\program.exe** files\sub dir\program name

**c:\program files\sub.exe** dir\program name

**c:\program files\sub dir\program.exe** name

**c:\program files\sub dir\program name.exe**

Si el módulo ejecutable es una aplicación de 16 bits, lpApplicationName debe ser NULL, y la cadena apuntada por lpCommandLine debe especificar el módulo ejecutable así como sus argumentos.

Para ejecutar un archivo por lotes, debe iniciar el intérprete de comandos; establezca lpApplicationName en cmd.exe y establezca lpCommandLine en los siguientes argumentos: / c más el nombre del archivo por lotes.

**lpCommandLine** [in, out, optional]

La línea de comando a ser ejecutada. La longitud máxima de esta cadena es 32.768 caracteres, incluido el carácter nulo de terminación Unicode. Si lpApplicationName es NULL, la porción del nombre del módulo de lpCommandLine está limitada a MAX\_PATH caracteres.

La versión Unicode de esta función, CreateProcessW, puede modificar el contenido de esta cadena. Por lo tanto, este parámetro no puede ser un puntero a la memoria de solo lectura (como una variable const o una cadena literal). Si este parámetro es una cadena constante, la función puede causar una violación de acceso.

El parámetro lpCommandLine puede ser NULL. En ese caso, la función usa la cadena apuntada por lpApplicationName como la línea de comando.

**lpProcessAttributes** [en, opcional]

Un puntero a una estructura SECURITY\_ATTRIBUTES que determina si el proceso devuelto al nuevo objeto de proceso puede ser heredado por procesos secundarios. Si lpProcessAttributes es NULL, el identificador no se puede heredar.

El miembro lpSecurityDescriptor de la estructura especifica un descriptor de seguridad para el nuevo proceso. Si lpProcessAttributes es NULL o lpSecurityDescriptor es NULL, el proceso obtiene un descriptor de seguridad predeterminado. Las ACL en el descriptor de seguridad predeterminado para un proceso provienen del token primario del creador.

Windows XP / 2000: las ACL en el descriptor de seguridad predeterminado para un proceso proviene del token primario o de suplantación del creador. Este comportamiento cambió con Windows XP con SP2 y Windows Server 2003.

**lpThreadAttributes** [en, opcional]

Un puntero a una estructura SECURITY\_ATTRIBUTES que determina si los procesos hijo pueden heredar el identificador devuelto al nuevo objeto thread. Si lpThreadAttributes es NULL, el identificador no se puede heredar.

**bherithandles** [in]

Si este parámetro es VERDADERO, el nuevo proceso heredará cada identificador heredable en el proceso de llamada. Si el parámetro es FALSE, los identificadores no se heredan. Tenga en cuenta que los identificadores heredados tienen el mismo valor y derechos de acceso que los identificadores originales.

Servicios de Terminal Server: no puede heredar identificadores en todas las sesiones. Además, si este parámetro es VERDADERO, debe crear el proceso en la misma sesión que la persona que llama.

Procesos de luz protegida (PPL): la herencia de identificador genérico se bloquea cuando un proceso de PPL crea un proceso que no es de PPL, ya que PROCESS\_DUP\_HANDLE no está permitido desde un proceso que no es de PPL hasta un proceso de PPL. Consulte Seguridad de proceso y derechos de acceso

**dwCreationFlags** [en]

Las banderas que controlan la clase de prioridad y la creación del proceso. Para obtener una lista de valores, consulte Indicadores de creación de procesos.

Este parámetro también controla la clase de prioridad del nuevo proceso, que se usa para determinar las prioridades de programación de los hilos del proceso. Si no se especifica ninguno de los indicadores de clase de prioridad, la clase de prioridad tiene como valor predeterminado NORMAL\_PRIORITY\_CLASS a menos que la clase de prioridad del proceso de creación sea IDLE\_PRIORITY\_CLASS o BELOW\_NORMAL\_PRIORITY\_CLASS. En este caso, el proceso hijo recibe la clase de prioridad predeterminada del proceso de llamada.

**ipEnvironment** [en, opcional]

Un puntero al bloque de entorno para el nuevo proceso. Si este parámetro es NULL, el nuevo proceso utiliza el entorno del proceso de llamada.

Un bloque de entorno consiste en un bloque terminado en nulo de cadenas terminadas en nulo. Cada cadena está en la siguiente forma:

nombre = valor \ 0

Como el signo igual se usa como separador, no se debe usar en el nombre de una variable de entorno.

Un bloque de entorno puede contener caracteres Unicode o ANSI. Si el bloque de entorno señalado por lpEnvironment contiene caracteres Unicode, asegúrese de que dwCreationFlags incluya CREATE\_UNICODE\_ENVIRONMENT. Si este parámetro es NULL y el bloque de entorno del proceso principal contiene caracteres Unicode, también debe asegurarse de que dwCreationFlags incluya CREATE\_UNICODE\_ENVIRONMENT.

La versión ANSI de esta función, CreateProcessA falla si el tamaño total del bloque de entorno para el proceso supera los 32.767 caracteres.

Tenga en cuenta que un bloque de entorno ANSI termina en dos bytes cero: uno para la última cadena y otro para terminar el bloque. Un bloque de entorno Unicode termina en cuatro bytes cero: dos para la última cadena, dos más para terminar el bloque.

**lpCurrentDirectory** [en, opcional]

La ruta completa al directorio actual para el proceso. La cadena también puede especificar una ruta UNC.

Si este parámetro es NULL, el nuevo proceso tendrá la misma unidad y directorio actuales que el proceso de llamada. (Esta función se proporciona principalmente para las shells que necesitan iniciar una aplicación y especificar su unidad inicial y su directorio de trabajo).

**Valor de retorno**

Si la función tiene éxito, el valor devuelto es distinto de cero.

Si la función falla, el valor de retorno es cero.

**Observaciones de seguridad**

El primer parámetro, lpApplicationName, puede ser NULL, en cuyo caso el nombre del ejecutable debe estar en la cadena delimitada por espacios en blanco apuntada por lpCommandLine. Si el ejecutable o la ruta de acceso tienen un espacio, existe el riesgo de que se ejecute un archivo ejecutable diferente debido a la forma en que la función analiza los espacios.

**Ejemplo CreateProcess**

C++

#include <windows.h>

#include <stdio.h>

#include <tchar.h>

void \_tmain( int argc, TCHAR \*argv[] )

{

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory( &si, sizeof(si) );

si.cb = sizeof(si);

ZeroMemory( &pi, sizeof(pi) );

if( argc != 2 )

{

printf("Usage: %s [cmdline]\n", argv[0]);

return;

}

// Start the child process.

if( !CreateProcess( NULL, // No module name (use command line)

argv[1], // Command line

NULL, // Process handle not inheritable

NULL, // Thread handle not inheritable

FALSE, // Set handle inheritance to FALSE

0, // No creation flags

NULL, // Use parent's environment block

NULL, // Use parent's starting directory

&si, // Pointer to STARTUPINFO structure

&pi ) // Pointer to PROCESS\_INFORMATION structure

)

{

printf( "CreateProcess failed (%d).\n", GetLastError() );

return;

}

// Wait until child process exits.

WaitForSingleObject( pi.hProcess, INFINITE );

// Close process and thread handles.

CloseHandle( pi.hProcess );

CloseHandle( pi.hThread );

}

\*Si CreateProcess tiene éxito, devuelve una estructura PROCESS\_INFORMATION que contiene controladores e identificadores para el nuevo proceso y su hilo principal. Los identificadores de subproceso y proceso se crean con derechos de acceso completos, aunque el acceso puede restringirse si especifica descriptores de seguridad.

**Ejemplo CreateProcess en TestShell**

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**#include <stdio.h>**

**#include <windows.h> /\*STARTUPINFO, PROCESS\_INFORMATION\*/**

**#include "Pol\_Directory.h"**

**#include "String\_Tokenizer.h"**

**#include "Polinomio.h"**

**#define MAXSIZE 1000**

**int last\_backslash(TCHAR\*);**

**void get\_substr(TCHAR\* cmdL,int intCantidad,char charBuf[MAXSIZE]);**

**#define TEST\_SHELL**

**#ifdef TEST\_SHELL**

**int main(){**

**TCHAR\* cmdline;**

**STARTUPINFO startup\_info;**

**PROCESS\_INFORMATION process\_info;**

**char charBuffer[MAXSIZE];**

**char exepath[MAXSIZE];**

**char str[MAXSIZE];**

**cmdline = GetCommandLine(); /\*obtiene la orden o comando equivalente de CLI\*/**

**/\*CLI: Command Line Interface\*/**

**cout<<cmdline<<endl;**

**int intCantidad=last\_backslash(cmdline);**

**printf("intCantidad=%i\n",intCantidad);**

**get\_substr(cmdline,intCantidad,charBuffer);**

**printf("%s\n",charBuffer);**

**char workspace[]="workspace.txt";**

**sprintf(exepath,"%s%s",charBuffer,"comando0.exe");**

**char dummy[]="dummy";**

**sprintf(str,"%s ./%s %s %s %s %s","comando0.exe",**

**workspace,"var1","+","var2",dummy);**

**printf("IMPRIMIR COMANDO A EJECUTAR PASANDO ARGUMENTOS:\n");**

**printf("%s\n",str);**

**fill\_n( (char\*)&process\_info, sizeof( PROCESS\_INFORMATION ), 0 );**

**fill\_n( (char\*)&startup\_info, sizeof( STARTUPINFO ), 0 );**

**startup\_info.cb = sizeof( STARTUPINFO );**

**if (!CreateProcess(**

NULL, // No module name (use command line)

exepath, // Command line

NULL, // Process handle not inheritable

NULL, // Thread handle not inheritable

FALSE, // Set handle inheritance to FALSE

0, // No creation flags

NULL, // Use parent's environment block

NULL, // Use parent's starting directory

&startup\_info, // Pointer to STARTUPINFO structure

&process\_info // Pointer to PROCESS\_INFORMATION structure

**)){**

**return 2;**

**}**

**CloseHandle( process\_info.hThread );**

**// wait for the child program to terminate**

**WaitForSingleObject( process\_info.hProcess, INFINITE );**

**//Leer el resultado**

**string source\_name=string(workspace);**

**Pol\_Directory the\_directory;**

**the\_directory.load\_data(source\_name);**

**string coeffs0=the\_directory.lookup\_entry("ans");**

**String\_Tokenizer tokenizer0(coeffs0,":");**

**vector<string> arr0;**

**while(tokenizer0.has\_more\_tokens()){**

**arr0.push\_back(tokenizer0.next\_token());**

**}//end while()**

**int num,den; /\*numerador,denominador\*/**

**Rac \*RacPt0=new Rac[arr0.size()];**

**string stringIntNum,stringIntDen;**

**String\_Tokenizer tokenizerForRac;**

**for(int i=0;i<arr0.size();i++){**

**tokenizerForRac=String\_Tokenizer(arr0[i],"/");**

**stringIntNum=tokenizerForRac.next\_token();**

**num=atoi(stringIntNum.c\_str());**

**stringIntDen=tokenizerForRac.next\_token();**

**den=atoi(stringIntDen.c\_str());**

**\*(RacPt0+i)=Rac(num,den);**

**}**

**Polinomio PolResult0(arr0.size()-1,RacPt0);**

**cout<<PolResult0<<endl;**

**cout<<PolResult0.string\_show()<<endl;**

**return 0;**

**}//end main()**

**#endif**

**/\*\***

**\* Devuelve la cantidad de car\'acteres que hay que usar de cmdL**

**\* para incluir hasta el \'ultimo backslash (es decir, desde el**

**\* car\'acter 0 hasta el car\'acter que devuelve esta funci\'on)**

**\*/**

**int last\_backslash(TCHAR\* cmdL){**

**int r=0,i=0,intI=0;**

**while('\0'!=cmdL[i]){**

**if('\\'==cmdL[i]){**

**r++;**

**}**

**i++;**

**}/\*Now r is the amount of backslashes\*/**

**i=0;**

**while((intI<r)&&('\0'!=cmdL[i])){**

**if('\\'==cmdL[i]){**

**intI++;**

**}**

**i++;**

**}**

**return i;**

**}**

**void get\_substr(TCHAR\* cmdL,int intCantidad,char charBuf[MAXSIZE]){**

**int i=0;**

**while(i<intCantidad){**

**charBuf[i]=cmdL[i];**

**i++;**

**}**

**charBuf[i]='\0';**

**}**

**Error/Solución (TestShell)**

|  |  |
| --- | --- |
| **if (!CreateProcess(**  exepath, // No module name (use command line)  str, // Command line  NULL, // Process handle not inheritable  NULL, // Thread handle not inheritable  FALSE, // Set handle inheritance to FALSE  0, // No creation flags  NULL, // Use parent's environment block  NULL, // Use parent's starting directory  &startup\_info, // Pointer to STARTUPINFO structure  &process\_info // Pointer to PROCESS\_INFORMATION structure  **))** | **if (!CreateProcess(**  NULL, // No module name (use command line)  exepath, // Command line  NULL, // Process handle not inheritable  NULL, // Thread handle not inheritable  FALSE, // Set handle inheritance to FALSE  0, // No creation flags  NULL, // Use parent's environment block  NULL, // Use parent's starting directory  &startup\_info, // Pointer to STARTUPINFO structure  &process\_info // Pointer to PROCESS\_INFORMATION structure  **))** |

La solución al problema dentro de los sistemas operativos Windows 8/10 fue colocar un NULL en el parámetro lpApplicationName ya que siguiendo las condiciones de seguridad y de dirección, el nombre de modulo (limitado por un espacio en negro) debe ser el primer token blanco, en donde será colocado la dirección del programa que queremos ejecutar.

De misma manera el parámetro bherithandles en nuestro caso es considerado FALSE en vez de TRUE, debido a que los indicadores que pudiera heredar no están presentes en todas las sesiones, además si este fuera TRUE sería necesario crear el proceso en la misma sesión.

*Cibergrafias:*

[1] CreateProcess function

Publicado en: https://msdn.microsoft.com/en-us/library/windows/desktop/ms682512(v=vs.85).aspx

[2], Creating Processes

Publicado en: https://msdn.microsoft.com/en-us/library/windows/desktop/ms682425(v=vs.85).aspx