

Re: CreateProcessAsUser – Process starts then exits

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.kernel/2008-01/msg00081.htm>

- *From:* coder0xff <coder0xff@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Mon, 7 Jan 2008 19:15:02 -0800
-

what os are you running?:

Windows XP Media Center (must be compatible with vista, however).

What Kind of Process is it?:

At first, the program I was trying to start was a windows forms app I made, but after seeing that it wasn't working I tried it with notepad.exe. Notepad also starts, and immediately closes. I know that notepad is being created from the correct executable because process monitor shows the notepad icon next to its log event.

Does it interact with the Desktop?:

The target program is intended to interact with the desktop. More specifically is enumerates the desktop windows and sends the data to my service through a named pipe.

Please show how you create the process, what the process is that has been created. some code would be very useful,... :

The code for it is actually through a bunch of .net classes that encapsulate the win32 WTS/token/desktop, etc functions. I have flattened it below. I have tested the code below and the result is identical. Call the below function from a service and use process monitor with the process start and process exit operation included.

Thanks again.

```
using System;
using System.Text;
using System.Runtime.InteropServices;
```

```
namespace Test
{
    static public class Test
    {
        /* structs, enums, and external functions defined at end of code */
```

```
public static System.Diagnostics.Process StartProcessInSession(int
sessionID, String commandLine)
```

Re: CreateProcessAsUser – Process starts then exits

```
{
IntPtr userToken;
if (WTSQueryUserToken(sessionID, out userToken))
{
//note that WTSQueryUserToken only works when in context of
local system account with SE_TCB_NAME
IntPtr lpEnvironment;
if (CreateEnvironmentBlock(out lpEnvironment, userToken,
false))
{
StartupInfo si = new StartupInfo();
si.cb = Marshal.SizeOf(si);
si.lpDesktop = "winsta0\\default";
ProcessInformation pi;
if (CreateProcessAsUser(userToken, null, new
StringBuilder(commandLine), IntPtr.Zero, IntPtr.Zero, false,
CreationFlags.DETACHED_PROCESS, lpEnvironment, null, ref si, out pi))
{
CloseHandle(pi.hThread);
CloseHandle(pi.hProcess);
//context.Undo();
try
{
return
System.Diagnostics.Process.GetProcessById(pi.dwProcessId);
}
catch (ArgumentException e)
{
//The process ID couldn't be found – which is
what always happens because it has closed
return null;
}
}
else
{
int err = Marshal.GetLastWin32Error();
throw new System.ComponentModel.Win32Exception(err,
"Could not create process.\nWin32 error: " + err.ToString());
}
}
else
{
int err = Marshal.GetLastWin32Error();
throw new System.ComponentModel.Win32Exception(err,
"Could not create environment block.\nWin32 error: " + err.ToString());
}
}
else
{
int err =
System.Runtime.InteropServices.Marshal.GetLastWin32Error();
```

Re: CreateProcessAsUser – Process starts then exits

```
if (err == 1008) return null; //There is no token
throw new System.ComponentModel.Win32Exception(err, "Could
not get the user token from session " + sessionID.ToString() + " – Error: " +
err.ToString());
}
}
```

```
[DllImport("wtsapi32.dll", SetLastError = true)]
static extern bool WTSQueryUserToken(Int32 sessionId, out IntPtr
Token);
```

```
[DllImport("userenv.dll", SetLastError = true)]
static extern bool CreateEnvironmentBlock(out IntPtr lpEnvironment,
IntPtr hToken, bool bInherit);
```

```
[DllImport("advapi32.dll", SetLastError = true, CharSet =
CharSet.Auto)]
static extern bool CreateProcessAsUser(IntPtr hToken, String
lpApplicationName, [In] StringBuilder lpCommandLine, IntPtr /*to a
SecurityAttributes struct or null*/ lpProcessAttributes, IntPtr /*to a
SecurityAttributes struct or null*/ lpThreadAttributes, bool bInheritHandles,
CreationFlags creationFlags, IntPtr lpEnvironment, String lpCurrentDirectory,
ref StartupInfo lpStartupInfo, out ProcessInformation lpProcessInformation);
```

```
[DllImport("kernel32.dll", SetLastError = true)]
static extern bool CloseHandle(IntPtr hHandle);
```

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
struct StartupInfo
{
public Int32 cb;
public String lpReserved;
public String lpDesktop;
public String lpTitle;
public Int32 dwX;
public Int32 dwY;
public Int32 dwXSize;
public Int32 dwYSize;
public Int32 dwXCountChars;
public Int32 dwYCountChars;
public Int32 dwFillAttribute;
public Int32 dwFlags;
public Int16 wShowWindow;
public Int16 cbReserved2;
public IntPtr lpReserved2;
public IntPtr hStdInput;
public IntPtr hStdOutput;
public IntPtr hStdError;
}
```

```
[StructLayout(LayoutKind.Sequential)]
```

Re: CreateProcessAsUser – Process starts then exits

```
internal struct ProcessInformation
{
public IntPtr hProcess;
public IntPtr hThread;
public int dwProcessId;
public int dwThreadId;
}
```

```
/// <summary>
/// The following process creation flags are used by the
CreateProcess, CreateProcessAsUser, CreateProcessWithLogonW, and
CreateProcessWithTokenW functions. They can be specified in any combination,
except as noted.
/// </summary>
```

[Flags]

```
enum CreationFlags : int
```

```
{
```

```
/// <summary>
```

```
/// Not specified by MSDN???
```

```
/// </summary>
```

```
NONE = 0,
```

```
/// <summary>
```

```
/// The calling thread starts and debugs the new process and all
child processes created by the new process. It can receive all related debug
events using the WaitForDebugEvent function.
```

```
/// A process that uses DEBUG_PROCESS becomes the root of a
debugging chain. This continues until another process in the chain is created
with DEBUG_PROCESS.
```

```
/// If this flag is combined with DEBUG_ONLY_THIS_PROCESS, the
caller debugs only the new process, not any child processes.
```

```
/// </summary>
```

```
DEBUG_PROCESS = 0x00000001,
```

```
/// <summary>
```

```
/// The calling thread starts and debugs the new process. It can
receive all related debug events using the WaitForDebugEvent function.
```

```
/// </summary>
```

```
DEBUG_ONLY_THIS_PROCESS = 0x00000002,
```

```
/// <summary>
```

```
/// The primary thread of the new process is created in a
suspended state, and does not run until the ResumeThread function is called.
```

```
/// </summary>
```

```
CREATE_SUSPENDED = 0x00000004,
```

```
/// <summary>
```

```
/// For console processes, the new process does not inherit its
parent's console (the default). The new process can call the AllocConsole
function at a later time to create a console. For more information, see
```

Re: CreateProcessAsUser – Process starts then exits

Creation of a Console.

/// This value cannot be used with CREATE_NEW_CONSOLE.

/// </summary>

DETACHED_PROCESS = 0x00000008,

/// <summary>

/// The new process has a new console, instead of inheriting its parent's console (the default). For more information, see Creation of a Console.

/// This flag cannot be used with DETACHED_PROCESS.

/// </summary>

CREATE_NEW_CONSOLE = 0x00000010,

/// <summary>

/// The new process is the root process of a new process group.

The process group includes all processes that are descendants of this root process. The process identifier of the new process group is the same as the process identifier, which is returned in the lpProcessInformation parameter. Process groups are used by the GenerateConsoleCtrlEvent function to enable sending a CTRL+BREAK signal to a group of console processes.

/// If this flag is specified, CTRL+C signals will be disabled for all processes within the new process group.

/// This flag is ignored if specified with CREATE_NEW_CONSOLE.

/// </summary>

CREATE_NEW_PROCESS_GROUP = 0x00000200,

/// <summary>

/// If this flag is set, the environment block pointed to by

lpEnvironment uses Unicode characters. Otherwise, the environment block uses ANSI characters.

/// </summary>

CREATE_UNICODE_ENVIRONMENT = 0x00000400,

/// <summary>

/// This flag is valid only when starting a 16-bit Windows-based application. If set, the new process runs in a private Virtual DOS Machine (VDM). By default, all 16-bit Windows-based applications run as threads in a single, shared VDM. The advantage of running separately is that a crash only terminates the single VDM; any other programs running in distinct VDMs continue to function normally. Also, 16-bit Windows-based applications that are run in separate VDMs have separate input queues. That means that if one application stops responding momentarily, applications in separate VDMs continue to receive input. The disadvantage of running separately is that it takes significantly more memory to do so. You should use this flag only if the user requests that 16-bit applications should run in their own VDM.

/// </summary>

CREATE_SEPARATE_WOW_VDM = 0x00000800,

/// <summary>

/// The flag is valid only when starting a 16-bit Windows-based application. If the DefaultSeparateVDM switch in the Windows section of

Re: CreateProcessAsUser – Process starts then exits

WIN.INI is TRUE, this flag overrides the switch. The new process is run in the shared Virtual DOS Machine.

/// </summary>

CREATE_SHARED_WOW_VDM = 0x00001000,

/// <summary>

/// The process is to be run as a protected process. The system restricts access to protected processes and the threads of protected processes. For more information on how processes can interact with protected processes, see Process Security and Access Rights.

/// To activate a protected process, the binary must have a special signature. This signature is provided by Microsoft but not currently available for non-Microsoft binaries. There are currently four protected processes: media foundation, audio engine, Windows error reporting, and system. Components that load into these binaries must also be signed. Multimedia companies can leverage the first two protected processes. For more information, see Overview of the Protected Media Path.

/// Windows Server 2003 and Windows XP/2000: This value is not supported.

/// </summary>

CREATE_PROTECTED_PROCESS = 0x00040000,

/// <summary>

/// The process is created with extended startup information; the lpStartupInfo parameter specifies a STARTUPINFOEX structure.

/// Windows Server 2003 and Windows XP/2000: This value is not supported.

/// </summary>

EXTENDED_STARTUPINFO_PRESENT = 0x00080000,

/// <summary>

/// The child processes of a process associated with a job are not associated with the job.

/// If the calling process is not associated with a job, this constant has no effect. If the calling process is associated with a job, the job must set the JOB_OBJECT_LIMIT_BREAKAWAY_OK limit.

/// </summary>

CREATE_BREAKAWAY_FROM_JOB = 0x01000000,

/// <summary>

/// Allows the caller to execute a child process that bypasses the process restrictions that would normally be applied automatically to the process.

/// Windows 2000: This value is not supported.

/// </summary>

CREATE_PRESERVE_CODE_AUTHZ_LEVEL = 0x02000000,

/// <summary>

/// The new process does not inherit the error mode of the calling process. Instead, the new process gets the default error mode.

/// This feature is particularly useful for multi-threaded shell

Re: CreateProcessAsUser – Process starts then exits

applications that run with hard errors disabled.

/// The default behavior is for the new process to inherit the error mode of the caller. Setting this flag changes that default behavior.

/// </summary>

CREATE_DEFAULT_ERROR_MODE = 0x04000000,

/// <summary>

/// The process is a console application that is being run without a console window. Therefore, the console handle for the application is not set.

/// This flag is ignored if the application is not a console application, or if it is used with either CREATE_NEW_CONSOLE or DETACHED_PROCESS.

/// </summary>

CREATE_NO_WINDOW = 0x08000000,

}

}

}

.