Destroying all child processes (and grandchildren) when the parent exits

devblogs.microsoft.com/oldnewthing/20131209-00

December 9, 2013

Raymond Chen

Today's Little Program launches a child process and then just hangs around. If you terminate the parent process, then all the children (and grandchildren and great-grandchildren, you get the idea) are also terminated.

The tool for this is the Job Object. Specifically, we mark the job as "kill on job close" which causes all processes in the job to be terminated when the last handle to the job is closed.

We must therefore be careful not to allow this handle to be inherited, because that would create another handle that needs to be closed before the job is terminated. And of course we need to be careful not to close the handle unless we really do want to terminate the job.

```
#define STRICT
#include <windows.h>
BOOL CreateProcessInJob(
    HANDLE hJob,
    LPCTSTR lpApplicationName,
    LPTSTR lpCommandLine,
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandles,
    DWORD dwCreationFlags,
    LPVOID lpEnvironment,
    LPCTSTR lpCurrentDirectory,
    LPSTARTUPINFO lpStartupInfo,
    LPPROCESS_INFORMATION ppi)
{
    BOOL fRc = CreateProcess(
        lpApplicationName,
        lpCommandLine,
        lpProcessAttributes,
        lpThreadAttributes,
        bInheritHandles,
        dwCreationFlags | CREATE_SUSPENDED,
        lpEnvironment,
        lpCurrentDirectory,
        lpStartupInfo,
        ppi);
    if (fRc) {
        fRc = AssignProcessToJobObject(hJob, ppi->hProcess);
        if (fRc && !(dwCreationFlags & CREATE_SUSPENDED)) {
            fRc = ResumeThread(ppi->hThread) != (DWORD)-1;
        }
        if (!fRc) {
            TerminateProcess(ppi->hProcess, 0);
            CloseHandle(ppi->hProcess);
            CloseHandle(ppi->hThread);
            ppi->hProcess = ppi->hThread = nullptr;
        }
    }
    return fRc;
}
```

The `CreateProcessInJob` function simply creates a process suspended, adds it to a job, and then resumes the process if the original caller asked for a running process.

Let's take it for a spin.

```
int __cdecl main(int, char **)
{
 HANDLE hJob = CreateJobObject(nullptr, nullptr);
 JOBOBJECT_EXTENDED_LIMIT_INFORMATION info = { };
 info.BasicLimitInformation.LimitFlags =
                    JOB_OBJECT_LIMIT_KILL_ON_JOB_CLOSE;
 SetInformationJobObject(hJob,
       JobObjectExtendedLimitInformation,
       &info, sizeof(info));
 STARTUPINFO si = { sizeof(si) };
 PROCESS_INFORMATION pi;
 char szCommandLine[] = "taskmgr.exe";
 if (CreateProcessInJob(hJob,
     "C:\\Windows\\System32\\taskmgr.exe",
     szCommandLine,
     nullptr, nullptr, FALSE, 0,
     nullptr, nullptr, &si, &pi)) {
  CloseHandle(pi.hProcess);
  CloseHandle(pi.hThread);
 }
 Sleep(30 * 1000);
 CloseHandle(hJob);
 return 0;
}
```

After creating the job object, we set the "kill children on close" flag on the job. Then we launch Task Manager into the job and give you 30 seconds to do your business. After that, it's "Time's up, you lose!" and Task Manager and any processes you launched from Task Manager will go away.

Raymond Chen

Follow