A more direct and mistake-free way of creating a process in a job object

devblogs.microsoft.com/oldnewthing/20230209-00

February 9, 2023

Raymond Chen

The traditional way of putting a process in a job object is

- Create the process suspended.
- Use `AssignProcessToJobObject` to put the process in a job object.
- Resume the process's initial thread.

Creating the process suspended is important, because you don't want the process to start running before you put it into a job. It could use that head start to do things that the job object was intended to prevent!

A problem with this approach is that there is a risk of orphaning the process if you crash or are terminated after you create the process suspended, but before you can move the suspended process into a job object.

I'm assuming that you created the job object as "terminate on last handle close", so that once the process is in the job, an unexpected crash of your program will result in the closure of the job object handle, which will terminate all the processes in it. The problem is that there is a brief moment in time in which the process is outside the job object, and that's your danger window.[1]

[1] It sounds like a window so small it may not be worth trying to close, but we've gotten reports from customers who are seeing lots of abandoned suspended processes on their servers, and we suspect that they may be hitting exactly this window.

One solution is to use the `PROC_THREAD_ATTRIBUTE_PARENT_PROCESS` attribute to specify a process already in the job as the new process's parent. Since child processes go into the same job as the parent, this creates the new process directly in the job, thereby closing the window. Of course, this means that you need to have an anchor process in the job already, which puts you in a Catch-22 situation: How do you safely get the anchor process into the job?

But there's a simpler solution: Windows 10 added the `PROC_THREAD_ATTRIBUTE_JOB_LIST` attribute so you can specify exactly which jobs you want the process to be assigned to. The process is assigned sequentially to each of the job handles you pass, so make sure you pass the job handles in an order that satisfies the rules for adding a process to multiple job objects. The assignment happens before the process's initial thread is allowd to run, so there is no race window where the process can do something bad before the job object limits can take effect.

Here's a quick demonstration. As is typical of Little Programs, there is little to no error checking.

```
int main(int, char**)
{
  HANDLE job = CreateJobObject(nullptr, nullptr);

  SIZE_T size;
  InitializeProcThreadAttributeList(nullptr, 1, 0, &size);
  auto p = (PPROC_THREAD_ATTRIBUTE_LIST)new char[size];

  InitializeProcThreadAttributeList(p, 1, 0, &size);
  UpdateProcThreadAttribute(p, 0,
    PROC_THREAD_ATTRIBUTE_JOB_LIST,
    &job, sizeof(job),
    nullptr, nullptr);

  wchar_t cmd[] = L"C:\\Windows\\System32\\cmd.exe";
  STARTUPINFOEX siex = {};
  siex.lpAttributeList = p;
  siex.StartupInfo.cb = sizeof(siex);
  PROCESS_INFORMATION pi;

  CreateProcessW(cmd, cmd, nullptr, nullptr, FALSE,
    CREATE_NEW_CONSOLE | EXTENDED_STARTUPINFO_PRESENT,
    nullptr, nullptr, &siex.StartupInfo, &pi);

  // Verify that the process is indeed in the job object.
  BOOL isInJob;
  IsProcessInJob(pi.hProcess, job, &isInJob);
  assert(isInJob);

  CloseHandle(pi.hProcess);
  CloseHandle(pi.hThread);
  delete[] (char*)p;
  CloseHandle(job);
  return 0;
}
```