How can I detect if one of my helper processes is launching child processes?

devblogs.microsoft.com/oldnewthing/20250523-00/?p=111216

May 23, 2025

A customer's program has a plug-in model. They already run the plug-ins in a separate process, but they wanted to understand, among other things, whether any of those plug-ins in turn launch child processes of their own. This would help them evaluate ideas for improving their plug-in model and reach out to plug-in authors who may be affected. They asked for ideas on how they could instrument this.

One of the things they considered was patching the import address table for all of the `CreateProcess*` functions so they could intercept attempts to create a child process. Another thing they considered was using the existing Detours library. They asked which method was better.

Better is not to do either of these things.

Instead, you can put the plug-in host process in a job object, and then monitor the job object. Specifically, job objects will queue the `JOB_OBJECT_MSG_NEW_PROCESS` completion when a new process is created, and it will queue the `JOB_OBJECT_MSG_EXIT_PROCESS` or `JOB_OBJECT_MSG_ABNORMAL_EXIT_PROCESS` completion when a process exits. (The system uses the process exit code to decide whether an exit was abnormal.)

So let's demonstrate this. We start with the existing Little Program that creates a job object and listens for completions and listen for the additional completions.

```
#define UNICODE
#define _UNICODE
#define STRICT
#include <windows.h>
#include <stdio.h>
#include <atlbase.h>
#include <atlalloc.h>
#include <shlwapi.h>

int __cdecl wmain(int argc, PWSTR argv[])
{
 CHandle Job(CreateJobObject(nullptr, nullptr));
 if (!Job) {
  wprintf(L"CreateJobObject, error %d\n", GetLastError());
  return 0;
 }


 CHandle IOPort(CreateIoCompletionPort(INVALID_HANDLE_VALUE,
                                       nullptr, 0, 1));
 if (!IOPort) {
  wprintf(L"CreateIoCompletionPort, error %d\n",
          GetLastError());
  return 0;
 }


 JOBOBJECT_ASSOCIATE_COMPLETION_PORT Port;
 Port.CompletionKey = Job;
 Port.CompletionPort = IOPort;
 if (!SetInformationJobObject(Job,
       JobObjectAssociateCompletionPortInformation,
       &Port, sizeof(Port))) {
  wprintf(L"SetInformation, error %d\n", GetLastError());
  return 0;
 }


 PROCESS_INFORMATION ProcessInformation;
 STARTUPINFO StartupInfo = { sizeof(StartupInfo) };
 PWSTR CommandLine = PathGetArgs(GetCommandLine());


 if (!CreateProcess(nullptr, CommandLine, nullptr, nullptr,
                    FALSE, CREATE_SUSPENDED, nullptr, nullptr,
                    &StartupInfo, &ProcessInformation)) {
  wprintf(L"CreateProcess, error %d\n", GetLastError());
  return 0;
 }


 if (!AssignProcessToJobObject(Job,
         ProcessInformation.hProcess)) {
  wprintf(L"Assign, error %d\n", GetLastError());
  return 0;
 }
```

```
 ResumeThread(ProcessInformation.hThread);
 CloseHandle(ProcessInformation.hThread);
 CloseHandle(ProcessInformation.hProcess);


 DWORD CompletionCode;
 ULONG_PTR CompletionKey;
 LPOVERLAPPED Overlapped;

 while (GetQueuedCompletionStatus(IOPort, &CompletionCode,
          &CompletionKey, &Overlapped, INFINITE)) {
  if ((HANDLE)CompletionKey == Job) {
    if (CompletionCode == JOB_OBJECT_MSG_ACTIVE_PROCESS_ZERO) {
      break; // all processes have exited - done
    } else if (CompletionCode == JOB_OBJECT_MSG_NEW_PROCESS) {
      wprintf(L"Process %d created\n", PtrToInt(Overlapped));
    } else if (CompletionCode == JOB_OBJECT_MSG_EXIT_PROCESS) {
      wprintf(L"Process %d exited\n", PtrToInt(Overlapped));
    } else if (CompletionCode == JOB_OBJECT_MSG_ABNORMAL_NEW_PROCESS) {
      wprintf(L"Process %d exited abnormally\n", PtrToInt(Overlapped));
    }
  }
 }

 wprintf(L"All done\n");

 return 0;
}
```

The original program checked only for `JOB_OBJECT_MSG_ACTIVE_PROCESS_ZERO` to exit the loop, but we added handlers for the three process-create/exit completion codes. (These code do not exit the loop.)

Left as an exercise (for further diagnostics) is using the process ID to get information like the path to the process. Note that there is a race condition if the process is very short-lived and exits before you can get any information from it.