Why is CreateToolhelp32Snapshot returning incorrect parent process IDs all of a sudden?

devblogs.microsoft.com/oldnewthing/20150403-00

April 3, 2015

Raymond Chen

A customer reported a problem with the `CreateToolhelp32Snapshot` function.

> From a 32-bit process, the code uses `CreateToolhelp32Snapshot` and `Process32-First` / `Process32Next` to identify parent processes on a 64-bit version of Windows. Sporadically, we find that the `th32ParentProcessID` is invalid on Windows Server 2008. This code works fine on Windows Server 2003. Here's the relevant fragment:
>
> ```
> std::vector<int> getAllChildProcesses(int pidParent)
> {
>  std::vector<int> children;
>
>  HANDLE snapshot = CreateToolhelp32Snapshot(
>     TH32CS_SNAPPROCESS, 0);
>  if (snapshot != INVALID_HANDLE_VALUE) {
>   PROCESSENTRY32 entry;
>   entry.dwSize = sizeof(entry); // weird that this is necessary
>   if (Process32First(snapshot, &entry)) {
>    do {
>     if (entry.th32ParentProcessID == pidParent) {
>      children.push_back(processEntry.th32ProcessID);
>     } while (Process32Next(snapshot, &entry));
>   }
>   CloseHandle(snapshot);
>  }
>  return children;
> }
> ```

(The customer snuck another pseudo-question in a comment. Here's why it is necessary.)

One of my colleagues asked what exactly was "invalid" about the process IDs. (This is like the StackOverflow problem where somebody posts some code and says simply "It doesn't work".)

My colleague also pointed out that the `thParentProcessID` is simply a snapshot of the parent process ID at the time the child process was created. Since process IDs can be recycled, once the parent process exits, the process ID is left orphaned, and it may get

reassigned to another unrelated process. For example, consider this sequence of events:

- Process A creates Process B.
- Process A terminates, thereby releasing its ID for reuse.
- Process C is created.
- Process C reuses Process A's process ID.

At this point, Process B will have a `th32ParentProcessID` equal to Process A, but since the ID for Process A has been reused for Process C, it will also be equal to Process C, even though there is no meaningful relationship between processes B and C.

If Process B needs to rely on its parent process ID remaining assigned to that process (and not getting reassigned), it needs to maintain an open handle to the parent process. (To avoid race conditions, this should be provided by the parent itself.) An open process handle prevents the process object from being destroyed, and in turn that keeps the process ID valid.

There is another trick of checking the reported parent process's creation time and seeing if it is more recent than the child process's creation time. If so, then you are a victim of process ID reuse, and the true parent process is long gone. (This trick has its own issues. For example, you may not have sufficient access to obtain the parent process's creation time.)

After a few days, the customer liaison returned with information from the customer. It looks like all of the guidance and explanation provided by my colleague either never made it to the customer, or the customer simply ignored it.

> The customer wants to detect what child processes are spawned by a particular application, let's call it P. We built a special version with extra logging, and it shows that the `PROCESSENTRY32.th32ParentProcessID` for `wininit.exe` and `csrss.exe` were both 0x15C, which is P's process ID. This erroneous reporting occurs while P is still running and continues after P exits. Do you think it's possible that process 0x15C was used by some other process earlier?

Yes, that possible. That is, in fact, what my colleague was trying to explain.

It isn't clear why the customer is trying to track down all child processes of process P, but the way to do this is to create a job object and put process P in it. You can then call `QueryInformationJobObject` with `JobObjectBasicProcessIdList` to get the list of child processes.

Raymond Chen

Follow