Process – Termination

By Anand George

Terminating a Process (MSDN)

- Terminating a process has the following results:
  - Any remaining threads in the process are marked for termination.
  - Any resources allocated by the process are freed.
  - All kernel objects are closed.
  - The process code is removed from memory.
  - The process exit code is set.
  - The process object is signaled.

How Processes are Terminated (MSDN)

- A process executes until one of the following events occurs:
  - Any thread of the process calls the **ExitProcess** function. Note that some implementation of the C run-time library (CRT) call **ExitProcess** if the primary thread of the process returns.
  - The last thread of the process terminates.
  - Any thread calls the **TerminateProcess** function with a handle to the process.
  - For console processes, the default console control handler calls **ExitProcess** when the console receives a CTRL+C or CTRL+BREAK signal.
  - The user shuts down the system or logs off.

Where do main functions return value go?

- GetProcessExitCode.
- This function returns immediately. If the process has not terminated and the function succeeds, the status returned is **STILL_ACTIVE**. If the process has terminated and the function succeeds, the status returned is one of the following values:
- The exit value specified in the **ExitProcess** or **TerminateProcess** function.
- The return value from the main or WinMain function of the process.
- The exception value for an unhandled exception that caused the process to terminate.

What goes? what stays? when a process dies?

- I did an malloc
- I did an fopen/ fwrite
- I created window
- I created a registry key?
- I edited a DB.
- I posed a message in Facebook.
- I copied something from a notepad and killed the notepad before paste.

- Thumb Rule?
  - Anything in process address space (user mode)memory gets cleaned up.
  - Handles are closed and objects are cleaned up if required by kernel.
  - Anything in devices, inter-process communications, data copied to other process address space stays like disk, packets send via NIC, clipboard etc.
- Now you know why the window of a process disappear when you kill that process from the task manager.

What if some resource in memory remains even after the process Termination?

Every time you've heard someone say they saw a ghost, or an angel. Every story you've ever heard about vampires, werewolves, or aliens, is the system assimilating some program that's doing something they're not supposed to be doing. The Oracle

A case study

- A socket handle
- Process is terminated.
- Handle remain open.
- Netstat, tcpview all show unknown process.
- Subsequent calls to listen to same port fail as existing handle is not closed.
- Programmer was not explicitly calling close handle as it was listening socket.

Investigation

- Owner process has unlinked from the process list in kernel.
- Check for any suspicious IRP on /Device/AFD file object.
- Did a complete memory search especially in kernel for any hidden binary. (I felt like a malware behavior)
- Check of any suspicious threads.
- All looks good.

Now what ?

- Found this,
  - https://support.microsoft.com/en-us/kb/2577795
  - Kernel sockets leak on a multiprocessor computer that is running Windows Server 2008 R2 or Windows 7
  - Installed the hotfix to get out of the mystery.

Resisting termination

- Pending IRP.
- Suspend process
- Attached debugger.
- Lot of ways if you have driver in kernel.

Bottom line

- We really need to worry if a process is leaking resources (handles) or memory once it is terminated unless these are some kinda shared resource which are owned by other processes.

Demo

- Termination and Exit code.

Summary

- Process Termination
- Resource clean up.

Thank you