Writing Topic #1: Processes

Alex Hoffer

CS 444
April 2017
Spring 2017

Contents

1 Windows

[4] [5] [3] [2] [1] The Windows operating system has many similarities and differences to Linux, and since it is proprietary software, it is not prone to the robust open source development that Linux is famous for. Nevertheless, Windows systems programming is well documented, and with the help of several resources we can get a good picture of how Windows manages processes, threads, and CPU scheduling time.

1.1 Processes

Processes are the fundamental building blocks of any operating system, and Windows' processes contain a lot of nuances that make process management seem, to me, more complex than in Linux. The following subsections denote where in Windows processes I see similarities to Linux and where I think there are differences between the two. Through this course of explanation, we will get a good idea of how Windows manages processes and reasons why Windows manages processes like they do.

1.1.1 Similarities to Linux

(1)Each process contains one or more threads. (2)Similarities in what resources a process inherits, such as: (3) Use of the C programming language, including struct-oriented design patterns

1.1.2 Differences from Linux

Process Creation To create a new process in Windows, we must execute a *CreateProcess* call. To do this same function in Linux, we traditionally use a *fork*. The differences between these calls is indicative of how the two operating systems handle the parent and child relationship inherent in a process hierarchy. Specifically, *CreateProcess* takes 10 arguments, including (but not limited to) desired security attributes, creation flags, and environment. The *fork* system call, on the other hand, does not take any arguments. Another important difference in process creation between these two systems is that the *CreateProcess* call returns a *bool* that is set to true if the process and subsequent thread are successful in creation, and false otherwise. *Fork()*, on the other hand, returns an integer value that, if the process is successfully created, corresponds to the process's ID. Thus, we can see that with Windows, the act of specifying a process's attributes are quite important, while with Linux, the programmer's awareness of the process hierarchy is emphasized, since they can easily store and access process IDs in order to identify which process is running at a given time. On the other hand, Windows does not maintain the relationships between parent and child processes [1]. [1] goes on to claim that the reason behind this lack of interest in parent and child process relationships in Windows is because the act of forking in Linux is, in practice, dreadful when creating multithreaded programs. This is because the fork system call makes it so that the newly created child process is an exact replica of the parent process, retaining precise copies of the parent's threads and synchronization objects, making management between multiprocessors a nightmare.

Process Identification In Windows, processes can be identified by both handles AND process IDs, while in Linux, we can only identify a process by its ID, which we can access using the *getpid()* system call. With Windows, we can get a process handle by using the *OpenProcess* function, and get a process ID by using the *GetCurrentProcessId*. The *OpenProcess* function allows for you to enumerate that handle's access rights and contains the ability to set whether or not the process is inheritable. I think Windows includes this handle as well as the process ID in a strategy to make security a priority in the act of process management, since you can specify what a process can access through its handle. In this sense, the identity of a Windows process is a much more complex entity, since it can include a large quantity of information.

Process Termination

1.2 Threads

1.2.1 Similarities to Linux

(1)Similarities in what resources each thread in a process shares, such as: (2) [1] claims POSIX Pthreads, which are used prominently in the Linux operating system, provides similar features to Windows threads, but Windows allows for a "broader collection of functions". SUCH AS: (3) Similar enough with POSIX Pthreads in how threads are managed that there are open source Pthreads libraries available for Windows system developers.

1.2.2 Differences from Linux

1.3 CPU Scheduling

1.3.1 Similarities to Linux

1.3.2 Differences from Linux

2 FreeBSD

2.1 Processes

2.1.1 Similarities to Linux

2.1.2 Differences from Linux

2.2 Threads

2.2.1 Similarities to Linux

2.2.2 Differences from Linux

2.3 CPU Scheduling

2.3.1 Similarities to Linux

2.3.2 Differences from Linux

References

[1] Johnson M. Hart. *Windows System Programming (3rd Edition)*. Addison-Wesley Professional, 2004.

[2] Michael Kerrisk. *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*. No Starch Press, San Francisco, CA, USA, 1st edition, 2010.

[3] Marshall Kirk. McKusick, George V. Neville-Neil, and Robert N. M. Watson. *The design and implementation of the FreeBSD operating system*. Addison-Wesley/Pearson, 2015.

[4] Mark E. Russinovich, Alex Ionescu, and David A. Solomon. *Windows internals, Part 2*. Microsoft, 2012.

[5] Mark E. Russinovich, David A. Solomon, and Alex Ionescu. *Windows internals*. Microsoft Press, 2012.