Writing 1

Writing 1 CS444 Spring2018

Brian Huang

1 Introduction

In operating systems there are three important pieces that the whole operating system is dependant on. These are processes, threads and a scheduler. A process is simply a program or a set of instructions executing on the operating system. This is true for all operating systems, but the way they are implemented can be vastly different. Inside of these processes are things called threads. Threads handle the process' tasks and are used for executing the various instructions that the process needs to execute. In an operating system there can be many threads and processes, so a scheduler is needed to schedule which process or thread will run and at what time and for how long. Schedulers vary from operating system to operating system, but they all do the same job, assigning what will happen and when.

2 Process In Windows

A process in windows is a container for a set of resources for the process to use when executing their instructions. In this container there is a private virtual address space, a executable program instruction, a list of open handles to system resources such as semaphores, ports, and files, and a process ID. Each of these processes will also point to the process that created it or its parent. If a parent process is terminated, the child process will still point to the nonexistent parent. This does not pose a problem to Windows, as nothing relies on this information, so having a child with a nonexistent parent does not cause any issues to the system.

3 In Comparison To Linux

Windows and linux differ in the fact that windows does not really keep track of parent and child. A child process in windows can still exist even after it's parent is terminated, but in linux terminating a parent with and still having the child exist can cause a zombie or orphan process, which could cause problems in the long run.

4 Process In FreeBSD

A process in FreeBSD has an address space containing a mapping of its program's object code and it's global variables. Processes in FreeBSD also have a set of kernel resources. These resources include credentials, signal state and any descriptor array that gives it access to files, pipes sockets and other devices. A process in FreeBSD must have system resources such as memory and the underlying CPU.

5 In Comparison To Linux

Processes in FreeBSD are pretty similar to those in Linux. Both of them stem from a central process using the fork() and exec() commands. This means that both Linux and FreeBSD have the same structure of parent, child, orphaned, and zombie processes. They even have similar ways of assigning PID's for each process.

6 Threads In Windows

A thread in windows is an entity within the process that is scheduled for execution. Without a thread in a process, nothing will happen. In Windows a thread has the contents of a set of CPU registers to represent the state of the processor, two stacks one for kernel mode and the other for user mode, a private storage area, and a unique identifier. These are some essential components of a windows thread. All threads within the same process shares the same virtual address space, which means all the threads in the same process will all share the same resources given to that process. What this also means is that all threads in that process have full read-write access to the process' virtual address space.

7 In Comparison To Linux

Threads in linux are unique in the fact that there are no threads, only processes. In linux there is nothing special about a thread and no special data structures. Threads are just standard processes that share resources with other processes. Each "thread" is unique to the kernel, but they processes that share their resources and address space with other processes. The main difference between Windows and Linux is this uniqueness. The difference of these two is that one processor would be assigned to the windows process with threads, but in linux one processor would be assigned to each of the threads, as they are not actually threads but they are processes.

8 Threads In FreeBSD

Threads in FreeBSD have a structure that contains a scheduling list which in turn contains the thread priority, scheduling priority, the amount of time spent sleeping and status flags. It also contains TSB which is the user and kernel mode execution states, the kernel stack and the machine state.

9 In Comparison To Linux

Threads in Linux and FreeBSD are implemented similarly where they are both implemented in a heavyweight fashion. What it means to be heavyweight is that instead of creating threads of a process another process is created instead. These processes essentially just share the same resources, but each one has its own processor to complete its task. This is different from windows which uses a lightweight system.

10 Scheduling In Windows

Scheduling in windows is priority driven, meaning that the highest priority process runs first. There are 32 priorities in windows ranging from 0 to 31. Real time levels have the priority level of 16 - 31, variable levels have the priority of 15 - 0, and 0 is reserved for the zero page thread. Processes also have processor affinity where the processor can limit a process' ability to to run. These affinities can be altered by altering the API or by setting an affinity mask. A process is given a quantum to run, which is just a length of time that a process can run, then another process of the same priority level will the run after the quantum is over. Scheduling is implemented at the kernel level, so there is no specific module that handles scheduling. The code that handles scheduling is spread all throughout the kernel.

11 In Comparison To Linux

The scheduling system in linux is fairly similar to that of windows. The scheduler is priority driven, however the scale in linux runs from -20 to +19 with a default priority level of zero. In Linux these are called "nice" values, where -20 is the highest priority while +19 is the lowest. Linux also has a second range for each process for the real-time priority. This ranges from 0 to 99 and acts opposite from the nice values, where 99 is the highest priority and 0 is the lowest. Both of these operating systems use a round-robin style to execute their processes. They first execute all their high priority processes, then move to a lower level of priority and execute all of those.

12 Scheduling In FreeBSD

A programs priority in FreeBSD is characterized according to the amount of computation and the amount of I/O the process will be doing. When a child process is created is shares the same priority level as the parent. The priority for a process is periodically recalculated based on the amount of CPU time that it has used and the amount of memory resources it requires for execution times. Real time processes are processes that have a deadline, and in FreeBSD there is a completely separate queue for these. This queue is completely unaffected by the regular time shared processes, and priorities for real time processes do not change or degrade. FreeBSD also has an idle priority, which is simply the lowest priority and only runs when it is the only thing left, or there are no more runnable processes available. Time shared processes also use a priority for scheduling, but there is a bias for interactive programs such as a word processor.

13 In Comparison To Linux

The scheduler in Linux and FreeBSD are fairly similar, where they both use a priority queue and another queue for real time processes. The main difference is that priorities in Linux are not biased or ever recalculated. They usually remain static, where as in FreeBSD there is a bias towards interactive programs and priorities are periodically recalculated.

14 Conclusion

Of these three operating systems Windows differs the most from FreeBSD and Linux, as FreeBSD and Linux are somewhat like relatives. They share similar process creation and similar thread implementation. One large difference of these three operating systems is the fact that windows can have a child without a parent. In FreeBSD and Linux this could eventually cause major problems, but in Windows it is completely fine as nothing relies on that information.

References

[1] A. L. Mark Russinovich, David A. Solomon, "Windows internals part 1," Microsoft Press, 2012, (Accessed on 4/18/2018).

[2] R. Love, "Linux kernel development," Pearson Education, 2010, (Accessed on 4/18/2018).

[3] R. N. W. Marshall Kirk Mckusick, George V. Neville-Neil, "The design and implementation of the freebsd operating system," pearson Education, 2015, (Accessed on 4/18/2018).