Concurrency Comparison

Operating Systems II

October 24, 2018

Author:

Erin Tanaka

1 LINUX

1.1 Processes

Processes are programs in execution. In Linux there are two different types of processes: foreground or interactive processes and background or non-interactive/automatic processes. Foreground processes are programs currently executing that the user interacts with. Background processes refer to programs executing but are not being interacted with by a user.

Each process is represented by a unique identifier and a task_struct data structure. The task vector points to every task_struct in the system. Linux processes each have unique numbers that identify them called process identifiers or PID. The init process is the parent of all processes on the system. It runs when the system boots up and manages all other processes on the system. The init process always has a PID of 1.

Linux processes can be in one of five states. The process state code R indicates that the process is running or runnable. If it is the current process being served by the CPU or if the process has all of they resources necessary to run but lacks CPU access respectively. S indicates that the process is currently sleeping. Sleeping processes are waiting for the resources to necessary to run. Once the resource the sleeping process is waiting on is available, the process wakes up and becomes runnable. Additionally, the process can be awoken early and become runnable if it receives a signal. Process state code D indicates a state identical to to that of S. The only difference is that it does not wake up and become runnable if it receives a signal. Stopped processes are indicated by process state code T. These processes have stopped executing. They are not running nor are they eligible to run. This state usually occurs when a process is stopped using a signal or it is being debugged. Defunct or zombie processes are indicated by process state code Z. These processes are dead and have been halted. However, they still have entry in the process table.

The top Linux command displays a real-time view to the processes currently running and being managed by the kernel. The command ps provides a static view of the current processes.

1.2 Threads

Linux implements threads the same as processes. To the kernel a thread is a process that shares resources with other processes. Each thread has a unique task_struct. Threads are created like normal tasks except the clone system call is passed flags that correspond to specific resources to be shared. In Linux you would have n processes which results in n normal task structs and some of the processes are set up to share specified resources.

The following command results in behavior identical to a normal fork with the exception that the address space, filesystem resources file descriptors and signal Handlers are shared.

```
clone (CLONE_VM | CLONE_FS | CLONE_FILES | CLONE_SIGHAND, 0);
```

This new task and its parent are considered threads. The various flags provided to the CLONE() help specify the new processes behavior and what resources the parent and child will be sharing.[1]

1.3 CPU Scheduling

The Linux scheduler utilizes different algorithms to schedule different kind of processes. The scheduler is modular in this sense. This method of various algorithms matching up with different processes is called scheduler classes. Each of the scheduler classes has a priority. Whichever scheduler is of the highest priority and has a runnable process will select which process runs next. The schedulers must account for the amount of time that a process will run so that it only runs a fair share of the processor.

2 WINDOWS

2.1 Processes

A Windows process is a container for a set of resources used when executing the instance of a program [2]. Windows processes are represented by and executive process (EPROCESS) structure. Like Linux, processes have their own independent virtual address space and unique process IDs. Each process points to its parent or the creator process. In Windows it is possible for processes to point to parent processes that do not exist anymore. There are several methods of viewing Windows processes. One of the most popular ways it the Task Manager.

2.2 Threads

Threads are entities withing processes that get scheduled for execution. Windows threads are represented by executive thread objects. Windows threads consist of contents of a set of CPU registers representing the state of the processor, two stacks for executing in kernel mode and user mode, a thread-local storage, a thread ID and sometimes their own security tokens [2]. A windows thread is represented by and executive thread object. Threads can be in several different states: ready, deferred ready, standby, running, waiting transition, terminated, and initialized [2].

2.3 CPU Scheduling

The Windows scheduling system is priority based. This means that a thread with one of the highest priorities and is runnable always runs. However, some high priority threads may not be able to run because they may not have access to the processors they are allowed to run on. Once the scheduler selects a thread to run, the thread runs for a given amount of time. Windows uses 32 priority levels to rank threads [2].

3 FreeBSD

3.1 Processes

Similar to linux, FreeBSD processes are programs that are in execution.They have an address space, a set of kernel resources, and at least one thread that executes its code.FreeBSD processes each have a unique process identifier (PID) like Linux processes. FreeBSD processes keep track of their own PID and their parent process' PID. Processes can be in the following states: NEW, NORMAL, or ZOMBIE. Processes are marked NEW when they are created with the fork system call. they are in the NORMAL state when they have all of the necessary resources allocated for a it to begin executing. Processes stay in the NORMAL state until they terminate. Processes are in the ZOMBIE state if it is deceased until it frees its resources and tells its parent process it has terminated [3].

3.2 Threads

A FreeBSD thread is the unit of execution in a process. They need an address space and various other resources. A thread can care its resources with other threads. Threads represent a virtual processor with a full context with of register state and their own stack mapped to the address space[3]. Threads have corresponding kernel threads that have individual kernel stacks to represent the user thread [3]. Threads switch between RUNNABLE, SLEEPING, and STOPPED states. In FreeBSD, a kernel thread will be awakened by a signal only if it sets the PCATCH flag when it sleeps[3]. This is similar to Linux processes with state codes S and D.

3.3 CPU Scheduling

Threads get CPU time based on their scheduling class and their scheduling priority [3]. The thread with the highest priority class will get run by the kernel. Priorities are set using the rtprio system call. Once set, they do not get changed by the kernel. Threads are assigned two priorities, one for user-mode execution and one for kernel-mode execution [3].

References

[1] R. Love, *Linux Kernel Development: A thorough guide to the design and implementation of the Linux kernel*. Crawfordsville, IN: Pearson Education Inc, 2010.

[2] M. Russinovich, D. A. Solomon, and A. Ionescu, *Windows Internals: Part one*. Redmond, WA: Microsoft Press, 2012.

[3] M. K. McKusick, G. V. Neville-Neil, and R. N. M. Watson, *The Design and Implementation of the FreeBSD Operating System*. Westford, MA: Pearson Education, 2015.