# Writing #1
# Scheduling, Processes, Threads

Oregon State University

CS444 Operating System II

Spring 2018

**Erich Kramer**

April 18, 2018

# 1 Linux

## 1.1 Processes

Linux processes exist in memory and have related files resting under /proc so that programs may interact with other processes in a safe way. Here information about file descriptors, cwd, attributes and various other important process data can be parsed. Prior to this the solution was to allow executing programs access to the Kernel memory data structures [1]. This has both heavy over head and potential security holes.

Processes exist in a hierarchy that traces back to the init system. All processes have a parent which can be either the process which it descends from, or the init system itself. Typically a process which has the init system as its parent and was created after the boot process is a daemon[2]. In some systems the parent of the init system is PID 0, the scheduler. However this is inconsistant and an ambiguous phrasing as it becomes difficult to distinguish between a NUll value and a PID of 0.

Typically processes on Linux exist in a few possible states, first they are initialized and immedietly are runnable. When they are picked up by the scheduler and ran they may either exit if they have completed their task or they will enter the blocked state if they exceed their allocated CPU time. From there they proceed to runnable ad. nasuem until their termination. However, before they can be cleaned up they are in a zombie state in which a parent must wait on them to die. [3]

## 1.2 Threads

Prior to Linux 2.4 threads had unique PIDs and were implemented as processes. The main difference between them and another system process was that they shared an executable and memory. They were still distinct from fork()'d processes.

However, after Linux2.4 thread groups were added as a feature[4]. In conjunction with the clone() function threads no longer have seperates PIDs.

## 1.3 Scheduling

### 1.3.1 CFS Scheduler

The Linux Kernel uses an algorithm called the "Completely Fair Scheduler" It is the successor to the O(1) Scheduler, which boasted a linear time decision making. The CFS makes use of a red-black tree to keep track of used virtual time and assert fairness. The intent is not to assign CPU time in a round robin fashion equally to processes. It is instead to assign time proportional to the priority of a process[6]. In doing this the scheduler is able to assign time to low priority tasks despite while still giving the majority to those of a higher priority.

# 2 Windows

## 2.1 Processes

According to MSDN docs[7], processes are organized under applications, and each process may contain one or more thread. These processes have their own security context and environment, and inside this enviornment there exists a primary thread which instantiates execution. From this thread, similarly to the init process in Linux with regards to processes, spawns more threads inside the context of this application. This then allows an application to apply multiprocessing solutions and so on.

Of note is also the Job object which exists on windows. As processes manage threads that they contain, job objects also manage processes which they contain suggesting even more abstraction on windows [8].

## 2.2 Threads

As discussed above threads on windows exist for all applications [8]. Each thread in Windows maintains its own exception handlers, priority and thread local storage and identifier. As Windows seems to have a fetish for layers of abstraction on execution partitioning there also exists a layer underneath threads through the fiber concept that MS uses. They can be used as a substitute for threads and may be manually scheduled by the program.

## 2.3 Scheduling

Threads are managed via a scheduling priority value in the range 0-31. Threads which share a priority are treated the same by the system. Higher priority threads are run in round robin, dropping to lower priority onlyas necessary. If a lower priority thread is running and one of a higher value becomes runnable then the latter thread takes control for a time slice. [9] As intuition suggests this is not the best scheduling algorithm for a system and falls victim to low priority threads being given no execution time. This is not ideal as while you do want to prioritize some tasks you also want to have equality and all processes should be given CPU time. High load and high priority tasks even risk freezing a system.

# 3 FreeBSD

## 3.1 Processes

FreeBSD also uses Process IDs as does Linux. There are many similarities between the two kernels and their handling of processes as they are both direct Unix derivatives [12]. Similar to Linux processes are all spawned by the fork() command on one level or another. Often features implemented in either Linux or BSD end up becoming uniform between the two as there is heavy cross contamination.

### 3.2 Threads

As would be expected FreeBSD implements threads in ways similar to Linux as it sources itself in POSIX and UNIX specs. Whether the model uses a 1:1 or 1:N process to thread model is genearlly dependent on the release of BSD and the Kernel customization by users[13].

### 3.3 Scheduling

#### 3.3.1 ULE Scheduler

The ULE Scheduler is the successor to the BSD scheduler, and was introduced in BSD 5.0 [11]. It is currently the default scheduler due to advantages it has over the BSD Scheduler. Namely it overcame deficites in SMP and SMT. It is comprised of a High and Low level Scheduler. The Low level scheduler [14] reads from the runqueues and places threads into execution, while the high level scheduler operates every few seconds and arranges the threads to be executed[11].

This scheduler also has a unique Sleepqueue where a program is stored if it is blocking. The high level scheduler must operate on threads to move them into priorities.

## References


[1] *Filesystem Hierarchy Standard* The Linux Foundation

[2] Michael Kerrisk, *The Linux Programming Interface*
No Starch Press San Francisco, CA, USA 2010

[3] Kevin McGrath Operating Systems 2 Lecture, Oregon State University

[4] ikkachu "Are thread implemented as processes on Linux?" [Online]
https://unix.stackexchange.com/questions/364660/are-threads-implemented-as-processes-on-linux

[5] Himanshu Arora "Introduction to Linux Threads" [Online]
Available: https://www.thegeekstuff.com/2012/03/linux-threads-intro/?utm_source=tuicool

[6] Lie Ryan "Difference between Windows and Linux Scheduler" Access April 14th 2018 [Online] Available: https://superuser.com/questions/414604/difference-between-the-windows-and-linux-thread-scheduler

[7] "Processes and Threads" Microsoft Documentation [Online]
Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ms684841(v=vs.85).aspx

[8] "About Threads and Processes" Microsoft Documentation [Online]
Available: https://superuser.com/questions/414604/difference-between-the-windows-and-linux-thread-scheduler

[9] "Scheduling Priorities" Microsoft Documentation [Online]
Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ms685100(v=vs.85).aspx

[10] "Comparison of Solaris, Linux, and FreeBSD Kernels" Open Solaris [Online]
Available: https://web.archive.org/web/20080807124435/http://cn.opensolaris.org/files/solaris_linux_bsd_cmp.pdf

[11] "Process Management in the FreeBSD Operating System" informIT

[12] Processes and Daemons [Online]
Available: https://www.freebsd.org/doc/handbook/basics-processes.html

[13] "A look inside..." FreeBSD [Online]
Available: https://www.freebsd.org/doc/en_US.ISO8859-1/articles/linux-emulation/inside.html

[14] *ULE: A Modern Scheduler for FreeBSD* Jeff Roberson The FreeBSD Project September 2003