Writing Topic 1: Processes

Behnam Saeedi
CS444: Operating systems II

Spring 2017

Abstract

This document covers The implementation of processes, threads and CPU schedulers that are used in Linux, Windows and FreeBSD kernels. Initally it describes the Linux kernel and then it compares the Windows and FreeBSD kernel to Linux. Then it covers certain algorithms used in implementation. Finally it describes specific features of some of these implementations.

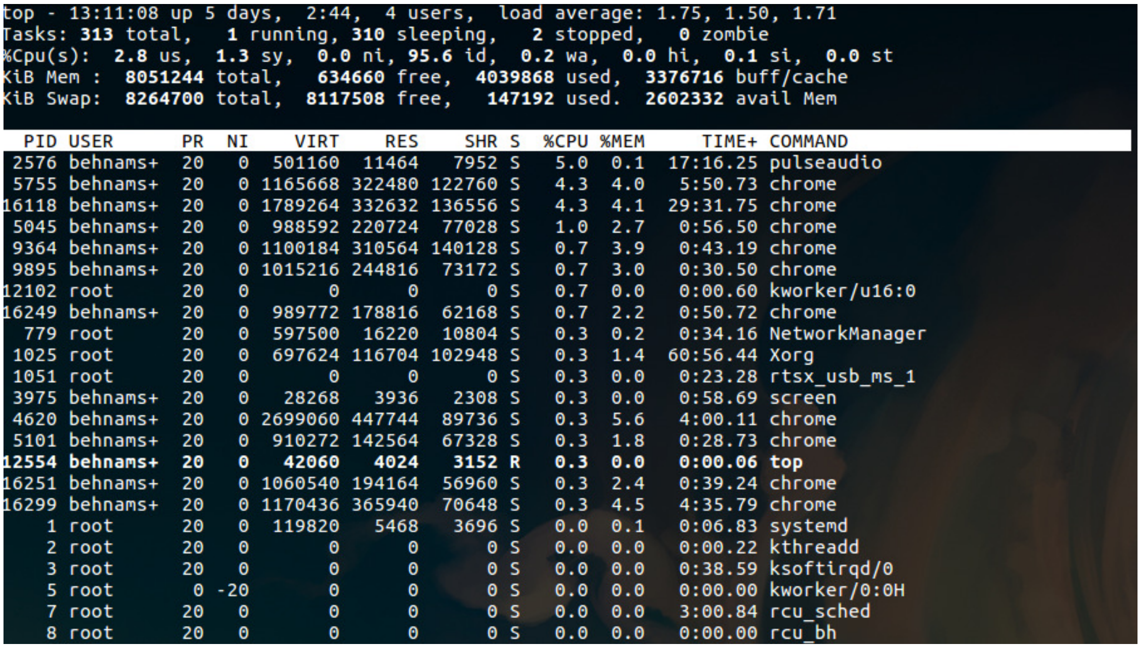
Contents

1 LINUX

1.1 Process

In Linux processes are instances of running programs. In linux the first process that is created with ID 1 is the init process. The processes could be viewed by two commands:

```
$ top
# and
$ ps
```



```
top - 13:11:08 up 5 days,  2:44,  4 users,  load average: 1.75, 1.50, 1.71
Tasks: 313 total,   1 running, 310 sleeping,   2 stopped,   0 zombie
%Cpu(s):  2.8 us,  1.3 sy,  0.0 ni, 95.6 id,  0.2 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem :  8051244 total,   634660 free,  4039868 used,  3376716 buff/cache
KiB Swap:  8264700 total,  8117508 free,   147192 used.  2602332 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 2576 behnams+  20   0  501160  11464   7952 S   5.0  0.1  17:16.25 pulseaudio
 5755 behnams+  20   0 1165668 322480 122760 S   4.3  4.0   5:50.73 chrome
16118 behnams+  20   0 1789264 332632 136556 S   4.3  4.1  29:31.75 chrome
 5045 behnams+  20   0  988592 220724  77028 S   1.0  2.7   0:56.50 chrome
 9364 behnams+  20   0 1100184 310564 140128 S   0.7  3.9   0:43.19 chrome
 9895 behnams+  20   0 1015216 244816  73172 S   0.7  3.0   0:30.50 chrome
12102 root      20   0       0      0      0 S   0.7  0.0   0:00.60 kworker/u16:0
16249 behnams+  20   0  989772 178816  62168 S   0.7  2.2   0:50.72 chrome
  779 root      20   0  597500  16220  10804 S   0.3  0.2   0:34.16 NetworkManager
 1025 root      20   0  697624 116704 102948 S   0.3  1.4  60:56.44 Xorg
 1051 root      20   0       0      0      0 S   0.3  0.0   0:23.28 rtsx_usb_ms_1
 3975 behnams+  20   0   28268   3936   2308 S   0.3  0.0   0:58.69 screen
 4620 behnams+  20   0 2699060 447744  89736 S   0.3  5.6   4:00.11 chrome
 5101 behnams+  20   0  910272 142564  67328 S   0.3  1.8   0:28.73 chrome
12554 behnams+  20   0   42060   4024   3152 R   0.3  0.0   0:00.06 top
16251 behnams+  20   0 1060540 194164  56960 S   0.3  2.4   0:39.24 chrome
16299 behnams+  20   0 1170436 365940  70648 S   0.3  4.5   4:35.79 chrome
    1 root      20   0  119820   5468   3696 S   0.0  0.1   0:06.83 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.22 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:38.59 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S   0.0  0.0   3:00.84 rcu_sched
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
```

Figure 1. The view of ”top” command in Linux shell

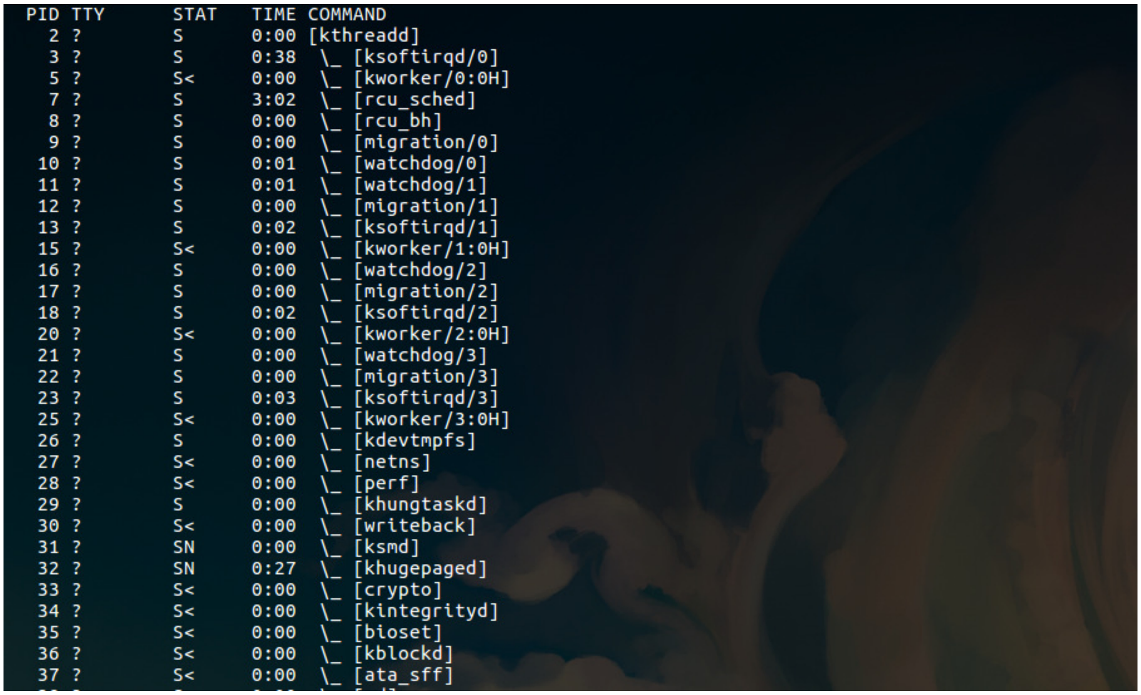
These two commands show a list of processes that are happening on the operating system. figures 1,2. top is a live view and ps is a static view of current processes. The init process could be found and viewed by the following command:

```
$ ps fax | grep [i]nit
    1 ?        Ss     0:06 /sbin/init splash
```

Each process in Linux could be in one of the following states:

- **Running:** Task is being processed or is about to be assigned to CPU for process.
- **Waiting:** interruptible and un-interruptible process waiting for resources to be assigned.
- **Stopped:** The process has been stopped by a signal. The process is in a halt.
- **Zombie :** This is a terminated process that is still consuming resources.

In Linux each process has an identifier and a task vectors. in order to see this feature better, Linux provides the ”pstree” command:



```
  PID TTY      STAT   TIME COMMAND
    2 ?        S      0:00 [kthreadd]
    3 ?        S      0:38  \_ [ksoftirqd/0]
    5 ?        S<     0:00  \_ [kworker/0:0H]
    7 ?        S      3:02  \_ [rcu_sched]
    8 ?        S      0:00  \_ [rcu_bh]
    9 ?        S      0:00  \_ [migration/0]
   10 ?        S      0:01  \_ [watchdog/0]
   11 ?        S      0:01  \_ [watchdog/1]
   12 ?        S      0:00  \_ [migration/1]
   13 ?        S      0:02  \_ [ksoftirqd/1]
   15 ?        S<     0:00  \_ [kworker/1:0H]
   16 ?        S      0:00  \_ [watchdog/2]
   17 ?        S      0:00  \_ [migration/2]
   18 ?        S      0:02  \_ [ksoftirqd/2]
   20 ?        S<     0:00  \_ [kworker/2:0H]
   21 ?        S      0:00  \_ [watchdog/3]
   22 ?        S      0:00  \_ [migration/3]
   23 ?        S      0:03  \_ [ksoftirqd/3]
   25 ?        S<     0:00  \_ [kworker/3:0H]
   26 ?        S      0:00  \_ [kdevtmpfs]
   27 ?        S<     0:00  \_ [netns]
   28 ?        S<     0:00  \_ [perf]
   29 ?        S      0:00  \_ [khungtaskd]
   30 ?        S<     0:00  \_ [writeback]
   31 ?        SN     0:00  \_ [ksmd]
   32 ?        SN     0:27  \_ [khugepaged]
   33 ?        S<     0:00  \_ [crypto]
   34 ?        S<     0:00  \_ [kintegrityd]
   35 ?        S<     0:00  \_ [bioset]
   36 ?        S<     0:00  \_ [kblockd]
   37 ?        S<     0:00  \_ [ata_sff]
```

Figure 2. The view of ”ps fax” command in shell

```
$ pstree
systemdModemManager{gdbus}
                      {gmain}
        NetworkManagerdhclient
                        dnsmasq
                        {gdbus}
                        {gmain}
        Xvfb{llvmpipe-0}
              {llvmpipe-1}
              {llvmpipe-2}
              {llvmpipe-3}
        accounts-daemon{gdbus}
                         {gmain}
        acpid
        agetty
        at-spi-bus-laundbus-daemon
                         {dconf worker}
                         {gdbus}
                         {gmain}
        at-spi2-registr{gdbus}
                         {gmain}
        avahi-daemonavahi-daemon
        bluetoothd
#                        .
#                        .
#                        .
```

Task vector is an array of pointers in a data structure sense that points to ever task struct data structure in the system. Furthermore, Unix introduced a system known as IPC or Inter-Process Communication. This allows different processes to communicate with one another.

1.2 Threads

From the kernel's perspective there is no distinction between threads and processes. Threads are series of processes that get access to the same memory space. From a CPU and scheduling perspective each thread gets its own time slot. We can trace and see each of these threads that are related to a process. Running the following code reveals an interesting fact about the Linux processes:

```
$ Htop
# or
$ top -H
# or
$ ps -T -p 1
```

init actually does not lunch any threads meaning that non of its children and the init itself would have access to the memory space of the child processes. Using the following command we can see the threads of each process:

```
$ ps -T -p $(ps efax | grep [m]y-app | awk '{print $1}')
# replace my-app with name of a program such as mysqld -> [m]ysqld
$ ps -T -p $(ps efax | grep [m]ysqld | awk '{print $1}')
```

```
  PID  SPID TTY          TIME CMD
 8302  8302 ?        00:00:00 mysqld
 8302  8307 ?        00:00:00 mysqld
 8302  8308 ?        00:00:03 mysqld
 8302  8309 ?        00:00:04 mysqld
 8302  8310 ?        00:00:03 mysqld
 8302  8311 ?        00:00:03 mysqld
 8302  8312 ?        00:00:03 mysqld
 8302  8313 ?        00:00:03 mysqld
 8302  8314 ?        00:00:03 mysqld
 8302  8315 ?        00:00:03 mysqld
 8302  8316 ?        00:00:03 mysqld
 8302  8317 ?        00:00:03 mysqld
 8302  8318 ?        00:00:03 mysqld
 8302  8320 ?        00:00:03 mysqld
 8302  8321 ?        00:00:05 mysqld
 8302  8322 ?        00:00:00 mysqld
 8302  8323 ?        00:00:04 mysqld
 8302  8324 ?        00:00:00 mysqld
 8302  8325 ?        00:00:00 mysqld
 8302  8326 ?        00:00:00 mysqld
 8302  8327 ?        00:00:00 mysqld
 8302  8328 ?        00:00:00 mysqld
 8302  8329 ?        00:00:00 mysqld
 8302  8330 ?        00:00:00 mysqld
 8302  8331 ?        00:00:00 mysqld
 8302  8332 ?        00:00:00 mysqld
 8302  8333 ?        00:00:00 mysqld
 8302  8336 ?        00:00:00 mysqld
```

All of these threads use the same address space as the mysqld process that was displayed in ps. In implementation, thread is actually implemented as a process. We can imagine process as a memory space and threads as units of execution of the program. many programs have a single threads for the process. Kernel to has a 1-1 ratio between both of them, it treats threads as full processes that share the same memory space. Even we try to spawn a new thread we use the same command as spawning a new process and pass different arguments to indicate if we want our new thread to be independent process or not.

1.3 CPU Scheduling

Linux uses complex data structures in order to become very efficient in binding processes to the CPU. Each processor gets its own run queue and selects processes from that specific run queue. This creates an issue where one processor ends up idling while there are processes on other processor's queue. In order to avoid this, the queues get rebalanced periodically. Each process gets a bitmask in order to identify which processor it can be processed on. This mask could be reassigned but re-balancing will not override affinity. The processor follows the following algorithm: [1][page 6]

- Highest priority queue with runnable processes is selected
- First process on that queue is selected
- Quantum size is calculated
- Runs until its time runs out
- its put on the expire queue
- the process repeats

As of 2007-04-13 a new patch put the CFS or completely fair scheduler in Linux kernel. [2]. As this memo mentions, a complete rewrite of the Linux task scheduler goes into place after this patch and replaces the O(1) scheduler that was used before. This scheduler follows the following algorithm: [3]

- The left most node of the scheduling tree is sent for execution.
- If completed, it is removed from the system and scheduling tree.
- If reached maximum execution time, it is stopped (voluntarily or via interrupt) and reinserted into the scheduling tree and a new execution time is calculated
- repeat

2 Windows

2.1 Process

In windows the processes are explained very similarly to Linux. A process is an executing program as described by MSDN. [4]. Windows also creates a new terminology called "Job Object". Job object allows multiple processes to be managed as a single unit. These object could have features assigned by OS such as namable, securable, etc.[4].

Similarly to Linux a process gets its own independent virtual address space. Each process contains one or more independently executing threads. Threads can create threads, processes and fibers. (Fibers are explained in Fibers subsection). It appears that in Windows, Process is a container that contains all of the threads. In other words, Process describes the address space which its threads will be using. Processes could be viewed and managed using several available tools.

2.1.1 Command: tasklist

Windows provides few tools in order to analyze the processes. The first tool was the tasklist. It provides a list of all of the tasks and processes that are running on the processor. Later, a Graphical user interface was developed for this command with the same name.

2.1.2 Close Program

The second tool that was introduced was called Close Program. This gave the user the ability of controlling the programs that were running at the time. This however did not last long since by introduction of multi processing it needed to change for more versatile tools.

2.1.3 Task Manager

The third tool is a graphical user interface designed to display the processes that are running on the system very similarly to the top command in Linux. This GUI is the Windows Task Manager.

2.2 Threads

Threads are also similar to Linux threads. The MSDN describes the threads as the basic unit which the kernel allocates time. A thread is part of a process that actually executed the programmed code. [4]. This functionality of a process still is fairly similar to the functionality of a thread in Linux with the difference that processes do not necessarily have to have threads in Linux where in windows each process must at least have one or more threads in order to execute code. From a code perspective, Windows has a completely different sets of syntax for creation and destruction of threads and processes.

In order to view threads in windows there aren't any tools provided that are shipped with the kernel for the computer user. However, there are third party programs that display the threads of each process. **Process Explorer** is a third-party tool that gives the user the ability to monitor different threads in windows processes.

2.3 Fiber

The MSDN document [4] introduces a concept called ”fiber”. Fiber is a unite of execution that is created by a thread and needs to be manually scheduled by the threads to be executed. [4]. Fibers share the thread context of their parent thread.

2.4 CPU Scheduling

Windows system scheduler controls the multitasking process by deciding which processing thread will recieve the next time slice. This process takes advantage of scheduling priorities.

2.4.1 Priorities

There are 6 possible processing priority classes for each process.

- Idle

- Bellow normal
- Normal
- High
- Realtime

Each priority class itself has 7 priority levels for each thread:

- Idle
- Lowest
- Bellow normal
- Normal
- Above normal
- Highest
- Time critical

The priority classes could be managed by the user in Task Manager. By default the priority of a process is normal. The priority levels on the other hand are assigned by the developer to each thread or is inherited from the parent process. This concept is described by the nice command in Linux. [5]. In windows the value of this priority is a number between 1 to 31 where 31 is the highest priority and 1 is Idle.

2.4.2 Context Switching

Similarly to linux the scheduler maintains a queue for ready tasks and priorities. Unlike Linux, the queue contains threads and their priorities only. The context switching happens in 5 steps:

- Save the context of the current thread
- Place the current thread at the end of the priority queue
- Find the next highest priority queue with ready threads
- remove the new thread, load its context and execute
- repeat

[6].

2.4.3 Algorithm

According to last sub section, windows uses a prioritized Round-Robin technique with multi-level feedback for priority handling. This technique has been the algorithm used for scheduling since NT. However, Vista did introduce some new heuristics in order to set priorities of certain processes. This scheduler is referred to as "Multi-level feedback queue" which still is a Round-Robin scheduler. Windows has also introduced a new concept to scheduling. This concept is referred to as User-mode Scheduling or UMS.[5]. This allows processes to schedule their own threads and we can see that in the fiber concept discussed earlier.

3 FreeBSD

3.1 Process

According to the documentation of freebsd, the processes are designed very similarly to Linux processes. The commands used for monitoring them are too same as Linux.[7]

3.2 Threads

The documentation for the freeBSD addresses 1:1 threading and 1:n Threading and describes the basics of the two threading methods. [8] Furthermore, it explains the libraries which implement threading in FreeBSD. It appears many of the features described in documentation are similar to Linux's implementation.

3.3 CPU Scheduling

FreeBSD Kernel has two available schedulers. The ULE scheduler first introduced in FreeBSD 5.0 and The 4.4BSD scheduler.

3.3.1 4.4BSD scheduler

This scheduler was originally designed for Linux kernel in order to address interactivity issues. This scheduler has a single run-queue and is variation of Round-Robin. This queue is in fact a double linked list and is shared accross all the CPUs. The reason to that is to avoid complex load balance algorithms.[9][Page 2].

3.3.2 ULE scheduler

ULE scheduler is based on the BSD schedulerand imporves on some of its features in order to improve efficiency specially with single processor systems. This scheduler is disabled by default in favor of the traditional BSD scheduler. Furthermore, it is fair by default but it could be set to favor specific processes.

References

[1] "cs.colombia.edu," https://www.cs.columbia.edu/~smb/classes/s06-4118/l13.pdf, accessed: 2017-05-01.

[2] "[patch] modular scheduler core and completely fair scheduler [cfs]," https://lwn.net/Articles/230501/, accessed: 2017-05-01.

[3] "Inside the linux 2.6 completely fair scheduler," https://www.ibm.com/developerworks/linux/library/l-completely-fair-scheduler/, accessed: 2017-05-01.

[4] "Processes and threads," https://msdn.microsoft.com/en-us/library/windows/desktop/ms684841(v=vs.85).aspx, accessed: 2017-05-01.

[5] "Scheduling," https://msdn.microsoft.com/en-us/library/windows/desktop/ms685096(v=vs.85).aspx, accessed: 2017-05-01.

[6] "Context switches," https://msdn.microsoft.com/en-us/library/windows/desktop/ms682105(v=vs.85).aspx, accessed: 2017-05-01.

[7] "Processes and daemons," https://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/basics-processes.html, accessed: 2017-05-01.

[8] "A look inside," https://www.freebsd.org/doc/en_US.ISO8859-1/articles/linux-emulation/inside.html, accessed: 2017-05-01.

[9] "Bfs by con kolivas," https://vellvisher.github.io/papers_reports/doc/BFS_FreeBSD.pdf, accessed: 2017-05-01.