DRAFT OF OUR TECHNICAL REPORT

Scheduling in Linux : CFS and it’s implementation

Fall 2019
Montana State University

Ellen Marie Andersen
Benjamin Cathelineau
Olexandr Matveyev

DRAFT OF OUR TECHNICAL REPORT

1 Brief history of CFS

The Completely Fair Scheduler was introduced in 2007, and it was merged into kernel-2.6.23 [6]. First shift from O(1) happened by Con Kolivas' development, with his Rotating Staircase Deadline Scheduler (RSDL). Later, Ingo Molnar developed CFS scheduler, and he incorporated some of the ideas from Con Kolivas' RSDL scheduler. Molnar had also developed the previously used O(1) scheduler [5].

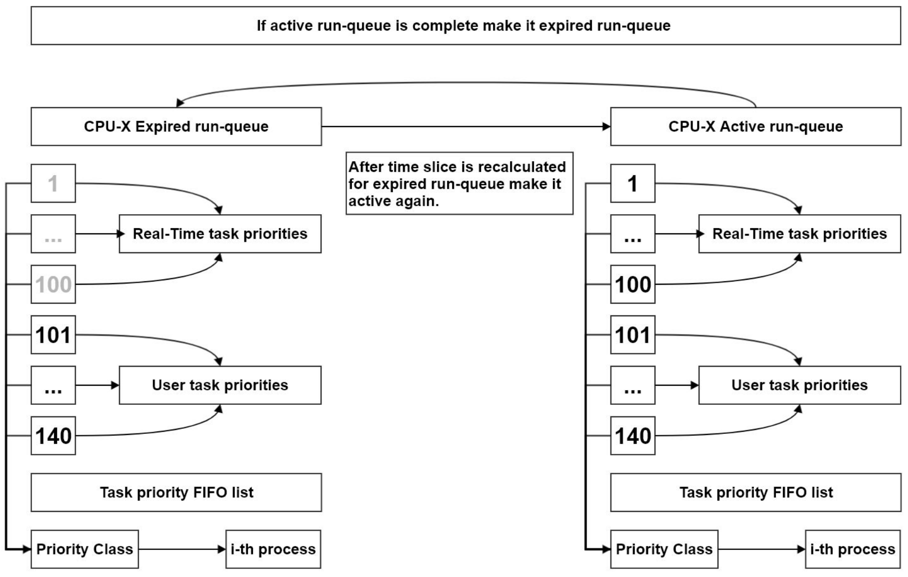
1.1 Reasons to change scheduler

It turned out that O(1) scheduler was very difficult to maintain due to the large amount of code and algorithm complexity ([3], pg: 462-463). The performance of interactivity with O(1) was poor, and it had low throughput for background jobs [2]. Interactive process requires short CPU bursts, and an example of interactive process is a text editor. The CFS provides and maintains balance by providing a fair amount of processor time to a task relative to other tasks [5].

1.2 Previous scheduler: O(1)

The O(1) scheduler is a constant time scheduler and it replaced O(n) scheduler. It was incorporated into Linux starting from kernel version 2.6.0 and till 2.6.22 version.In 2007 it was replaced by the Completely Fair Scheduler [1].

For the O(1) scheduling algorithm, in order to achieve constant scheduling time it must use active and expired arrays of processes. In this algorithm, each process gets a fixed time quantum. After time quantum expires for a task in the active array it will be placed into expired array, and array switch will take place. Due to that fact that arrays are accessed only via pointer, it makes array switching very fast O(1). When algorithm performs task switching it makes the active array as the new empty expired array, while the expired array becomes the active array [1].


If active run-queue is complete make it expired run-queue
CPU-X Expired run-queue
CPU-X Active run-queue
After time slice is recalculated for expired run-queue make it active again.
1
...
100
101
...
140
Real-Time task priorities
User task priorities
Task priority FIFO list
Priority Class
i-th process

Real-Time Priority tasks from 1 to 100 and Normal Priority tasks from 101 to 140 [4]

1.2.1 Shortcomings of the previous scheduler

The design goals of the Linux kernel scheduler version 2.6 O(1) is to achieve fairness and boost interactivity performance, but because O(1) did try to achieve fairness that actually resulted decreasing interactive performance [12]. The main reason to replace O(1) scheduler was the large mass of code needed to calculate heuristics, to mark a task as interactive or non-interactive which is making it fundamentally difficult to increase interactivity performance [5].

2 How CFS works : General presentation

2.1 Fair share Scheduling and what is it?

Let’s say we have 4 processes:

Process	**Running time**
Process A	3ms
Process B	2ms
Process C	4ms
Process D	3ms

How does Fair share scheduling work for these processes?

On the CPU, if only these four processes are to be executed and are ready, the processor will give each process a fair amount of CPU time. This means that the processes will alternate running on the CPU in fair time quantas while none of them are finished. This is calculated by N processes and (100/N)% of CPU time. At time 0, N is 4 and each process will get a fair share of the processor. This is shown in example figure below:

	2ms quanta, N = 4	Remaining running time	2ms quanta, N =4	Remaining running time	2ms quanta, N = 4	Remaining running time
Process A	0.5ms	2.5ms	0.5ms	2ms	0.5ms	1.5ms
Process B	0.5ms	1.5ms	0.5ms	1ms	0.5ms	0.5ms
Process C	0.5ms	3.5ms	0.5ms	3ms	0.5ms	2.5ms

Process D	0.5ms	2.5ms	0.5ms	2ms	0.5ms	1.5ms

continuing below:

	2ms quanta, N = 4	Remaining running time	2ms quanta, N =3	Remaining running time	2ms quanta, N = 3	Remaining running time
Process A	0.5ms	1ms	0.6667ms	0.3333ms	0.6667ms	0ms
Process B	0.5ms	0ms	n/a	n/a	n/a	n/a
Process C	0.5ms	3ms	0.6667ms	2.3333ms	0.6667ms	1.6667ms
Process D	0.5ms	1ms	0.6667ms	0.3333ms	0.6667ms	0ms

continuing below:

	2ms quanta, N = 1	Remaining running time	2ms quanta, N =1	Remaining running time
Process A	n/a	n/a	n/a	n/a
Process B	n/a	n/a	n/a	n/a
Process C	n/a	n/a	n/a	n/a
Process D	2ms	2ms	2ms	0ms

How does this differ if one of the processes have higher priority than another? Or if a process spawns many processes related to itself to capture more CPU resources? This is described in more detail below with virtual runtime and grouping scheduling [8].

2.2 The virtual runtime

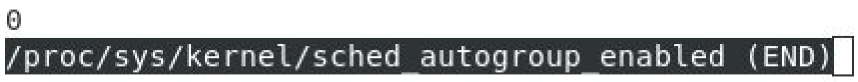
The method of the scheduler is simple, CFS tries to run the process with the lowest virtual runtime. Virtual runtime is calculated by t (time spent running) multiplied with the weight of the process' niceness/priority (explained below). As a process runs on the processor, t increases accordingly.

The virtual runtime is simply incremented differently according to the priority of the process. Processes with low priority number get their virtual runtime increased less. Process with high priority number gets their virtual runtime increased more.

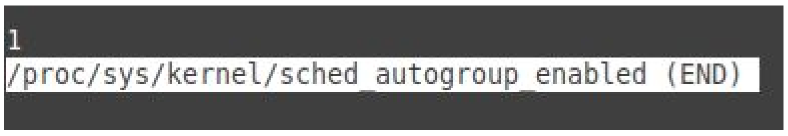
I/O bound processes vs Processor bound processes:

Niceness/priorities set aside, due to I/O processes typically executing at a short time interval each time before being blocked, these processes' vruntime will not increase a lot due to the short time bursts spent executing. If another process executes for double the time, the calculation will therefore be affected accordingly, and the process' place in the red black tree may likely be placed further to the right side of the tree. The further to the right side of the tree a process is placed, the less likely it is to get run next. The process on the leftmost and furthermost down side of the tree is the process with least vruntime and is picked to schedule next (see section 3.1 below). Starvation is combated in this model due to the 't' increase for each process, which ensures all processes are chosen to run. **how does this depend on niceness/priorities.**

In order for a process not to capture too much of the CPU by spawning new processes/tasks belonging to that process, Linux CFS can group certain processes/tasks that are related to each other and treat every group of tasks as one process. This is called group scheduling [9],([3], pg 464).
One can enable and disabling the grouping scheduling features in the scheduling algorithm in one's settings. Below shows that when 0 is returned, autogrouping is disabled and when 1 returns, autorouping is enabled.
Here is an example of the autogrouping (grouping scheduling) disabled and enabled.



```
0
/proc/sys/kernel/sched_autogroup_enabled (END)
```



```
1
/proc/sys/kernel/sched_autogroup_enabled (END)
```

2.3 Implementation of priorities

Priorities range from 0 to 139. The lower the number the higher the priority.
The user can customize priorities as explained later in this report

3 Analysis of source code

3.1 Data structure of CFS

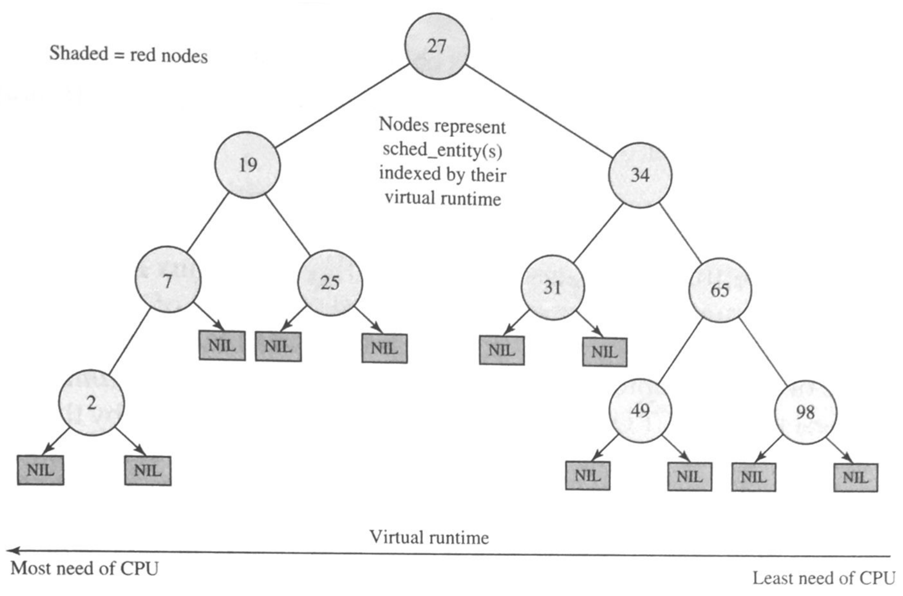


Figure 10.11 Example of Red Black Tree for CFS

RB Tree ([3]: pg 463)
The leftmost task is always picked by the scheduler. The RBTree is ordered by vruntime.

3.2 Algorithm of CFS

Based on the CFS documentation we can see that CFS has about 10000 lines of code [10].

4 The user side : What you can do to configure your scheduler in a Linux system :

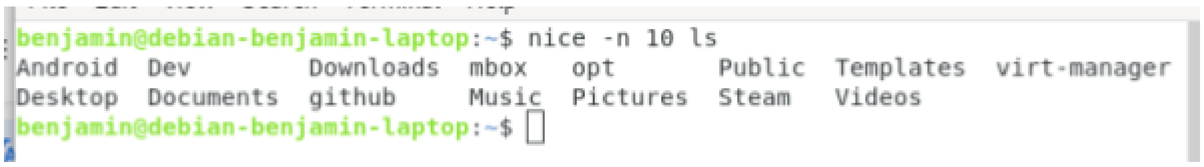
4.1 Niceness

On Linux, the user can customize the priority of a process, to a certain degree. This customisation is made through the use of niceness. The niceness of a process can range from -20 to 19. The lower the niceness, the higher the priority.
Only positive (lower) priority number are accessible to non root user. To set a negative priority you need to be root. However, like many things in Linux, this can be changed in the /etc/security/limits.conf file, to allow non privileged user to set negative niceness.[7]

Still, it is important to keep in mind that theses priority adjustment only concern non-real time process, as explained earlier in this report

4.1.1 nice

Nice is the standard utility to start a program with a different niceness than the default one (0).

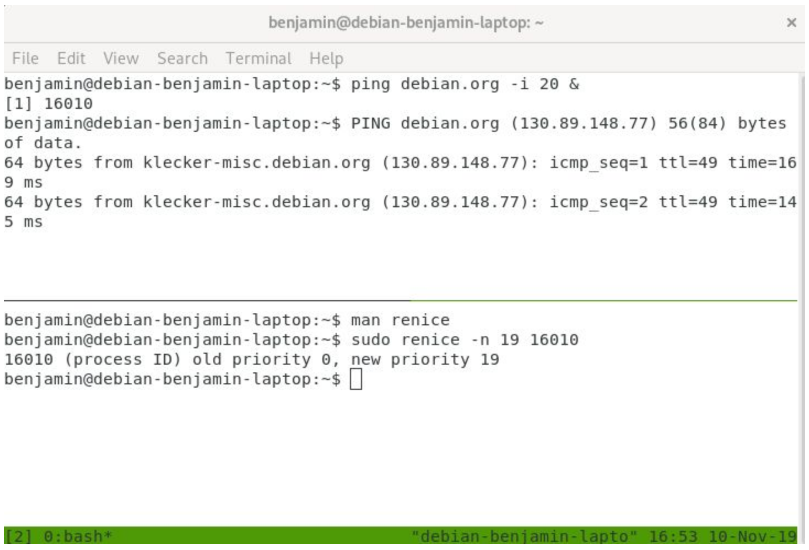


```
benjamin@debian-benjamin-laptop:~$ nice -n 10 ls
Android  Dev        Downloads  mbox   opt       Public  Templates  virt-manager
Desktop  Documents  github     Music  Pictures  Steam   Videos
benjamin@debian-benjamin-laptop:~$
```

Example usage of nice

4.1.2 renice

Renice is the utility that allows to change the niceness of an already started process.



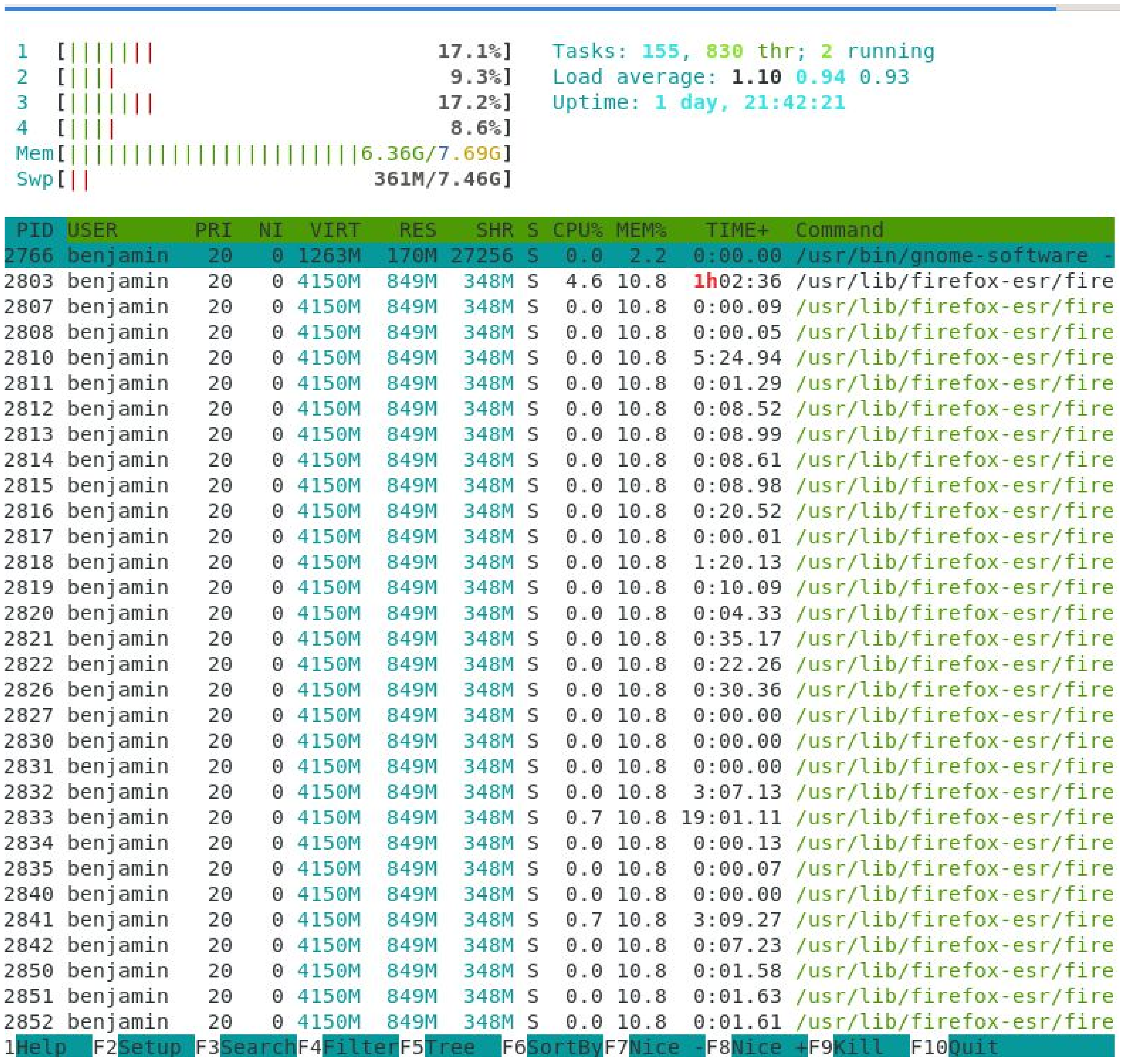
benjamin@debian-benjamin-laptop: ~
File Edit View Search Terminal Help

```
benjamin@debian-benjamin-laptop:~$ ping debian.org -i 20 &
[1] 16010
benjamin@debian-benjamin-laptop:~$ PING debian.org (130.89.148.77) 56(84) bytes
of data.
64 bytes from klecker-misc.debian.org (130.89.148.77): icmp_seq=1 ttl=49 time=16
9 ms
64 bytes from klecker-misc.debian.org (130.89.148.77): icmp_seq=2 ttl=49 time=14
5 ms

benjamin@debian-benjamin-laptop:~$ man renice
benjamin@debian-benjamin-laptop:~$ sudo renice -n 19 16010
16010 (process ID) old priority 0, new priority 19
benjamin@debian-benjamin-laptop:~$

[2] 0:bash*                              "debian-benjamin-lapto" 16:53 10-Nov-19
```

There is also a ionice program but it is outside the scope of this report because it's related to IO.

4.1.3 htop

HTOP is a general purpose command line task manager, inspired by top but better.



```
  1  [|||||||                     17.1%]   Tasks: 155, 830 thr; 2 running
  2  [||||                         9.3%]   Load average: 1.10 0.94 0.93
  3  [|||||||                     17.2%]   Uptime: 1 day, 21:42:21
  4  [||||                         8.6%]
  Mem[||||||||||||||||||||||6.36G/7.69G]
  Swp[||                     361M/7.46G]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 2766 benjamin   20   0 1263M  170M 27256 S  0.0  2.2  0:00.00 /usr/bin/gnome-software -
 2803 benjamin   20   0 4150M  849M  348M S  4.6 10.8  1h02:36 /usr/lib/firefox-esr/fire
 2807 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:00.09 /usr/lib/firefox-esr/fire
 2808 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:00.05 /usr/lib/firefox-esr/fire
 2810 benjamin   20   0 4150M  849M  348M S  0.0 10.8  5:24.94 /usr/lib/firefox-esr/fire
 2811 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:01.29 /usr/lib/firefox-esr/fire
 2812 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:08.52 /usr/lib/firefox-esr/fire
 2813 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:08.99 /usr/lib/firefox-esr/fire
 2814 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:08.61 /usr/lib/firefox-esr/fire
 2815 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:08.98 /usr/lib/firefox-esr/fire
 2816 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:20.52 /usr/lib/firefox-esr/fire
 2817 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:00.01 /usr/lib/firefox-esr/fire
 2818 benjamin   20   0 4150M  849M  348M S  0.0 10.8  1:20.13 /usr/lib/firefox-esr/fire
 2819 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:10.09 /usr/lib/firefox-esr/fire
 2820 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:04.33 /usr/lib/firefox-esr/fire
 2821 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:35.17 /usr/lib/firefox-esr/fire
 2822 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:22.26 /usr/lib/firefox-esr/fire
 2826 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:30.36 /usr/lib/firefox-esr/fire
 2827 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:00.00 /usr/lib/firefox-esr/fire
 2830 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:00.00 /usr/lib/firefox-esr/fire
 2831 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:00.00 /usr/lib/firefox-esr/fire
 2832 benjamin   20   0 4150M  849M  348M S  0.0 10.8  3:07.13 /usr/lib/firefox-esr/fire
 2833 benjamin   20   0 4150M  849M  348M S  0.7 10.8 19:01.11 /usr/lib/firefox-esr/fire
 2834 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:00.13 /usr/lib/firefox-esr/fire
 2835 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:00.07 /usr/lib/firefox-esr/fire
 2840 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:00.00 /usr/lib/firefox-esr/fire
 2841 benjamin   20   0 4150M  849M  348M S  0.7 10.8  3:09.27 /usr/lib/firefox-esr/fire
 2842 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:07.23 /usr/lib/firefox-esr/fire
 2850 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:01.58 /usr/lib/firefox-esr/fire
 2851 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:01.63 /usr/lib/firefox-esr/fire
 2852 benjamin   20   0 4150M  849M  348M S  0.0 10.8  0:01.61 /usr/lib/firefox-esr/fire
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

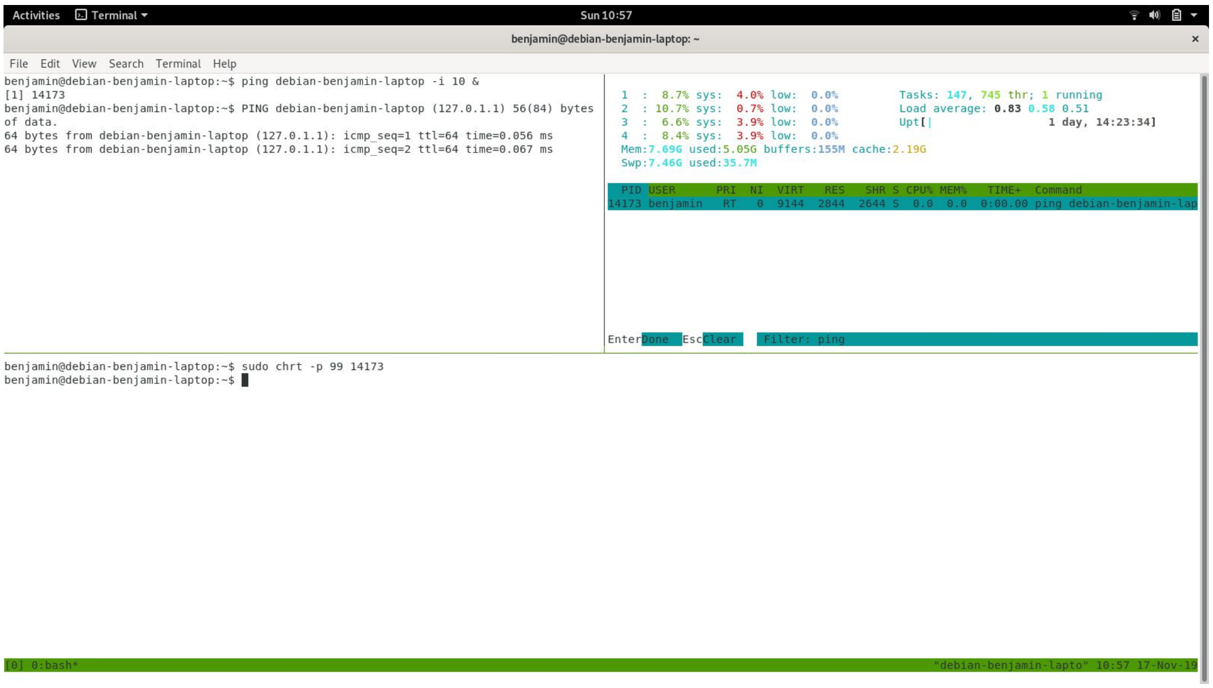
The priority can be increased (f7 decrease niceness) or decreased (f8 increase niceness). This tool is easier to use than renice because it displays the name of the program.

4.2 Real time attribute

4.2.1 chrt command

The default scheduling policies for processes is SCHED_OTHER, or SCHED_NORMAL (same thing). Processes under this policies are scheduled with CFS, and they use niceness. However they are other scheduling policies, such as SCHED_RR and SCHED_FIFO. Theses policies don’t use CFS and niceness. Instead they use static priorities. [14]
With chrt you can change the policy of your process, and it’s static priority.
Here is an example usage :

```
sudo chrt -p 99 10540
```



Activities
Terminal
Sun 10:57
benjamin@debian-benjamin-laptop: ~
File Edit View Search Terminal Help

```
benjamin@debian-benjamin-laptop:~$ ping debian-benjamin-laptop -i 10 &
[1] 14173
benjamin@debian-benjamin-laptop:~$ PING debian-benjamin-laptop (127.0.1.1) 56(84) bytes
of data.
64 bytes from debian-benjamin-laptop (127.0.1.1): icmp_seq=1 ttl=64 time=0.056 ms
64 bytes from debian-benjamin-laptop (127.0.1.1): icmp_seq=2 ttl=64 time=0.067 ms
```

```
  1  :  8.7% sys:  4.0% low:  0.0%        Tasks: 147, 745 thr; 1 running
  2  : 10.7% sys:  0.7% low:  0.0%        Load average: 0.83 0.58 0.51
  3  :  6.6% sys:  3.9% low:  0.0%        Upt[|                  1 day, 14:23:34]
  4  :  8.4% sys:  3.9% low:  0.0%
  Mem:7.69G used:5.05G buffers:155M cache:2.19G
  Swp:7.46G used:35.7M

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
14173 benjamin   RT   0  9144  2844  2644 S  0.0  0.0  0:00.00 ping debian-benjamin-lap

EnterDone  EscClear   Filter: ping
```

```
benjamin@debian-benjamin-laptop:~$ sudo chrt -p 99 14173
benjamin@debian-benjamin-laptop:~$ 
```

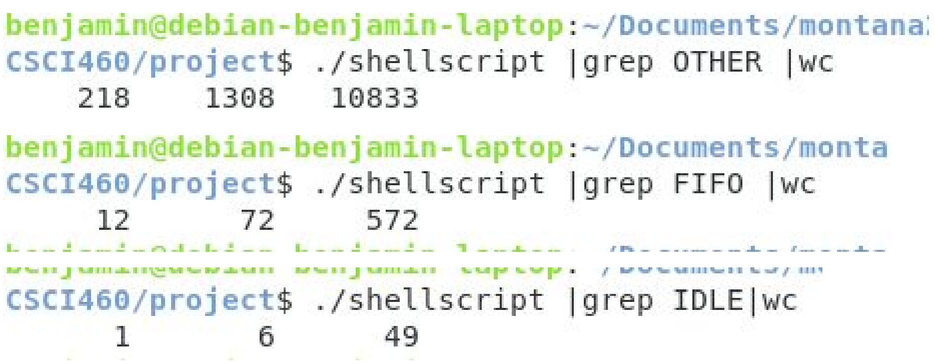
```
[0] 0:bash*                                          "debian-benjamin-lapto" 10:57 17-Nov-19
```

Here I did a ping and I changed it’s static priority to 99(highest possible priority).

4.2.2 Shell script with chrt command, to find which policies are the most common.

This shell script is heavily inspired and slightly modified, from[13]

```
#! /bin/bash
ps -aux |grep [0-9]|awk '{print $2}'  > test.txt
cat test.txt |while read line
do
chrt  -p $line 2> /dev/null
done
```



```
benjamin@debian-benjamin-laptop:~/Documents/montana
CSCI460/project$ ./shellscript |grep OTHER |wc
    218    1308   10833
benjamin@debian-benjamin-laptop:~/Documents/monta
CSCI460/project$ ./shellscript |grep FIFO |wc
     12      72     572
benjamin@debian-benjamin-laptop:~/Documents/mo
CSCI460/project$ ./shellscript |grep IDLE|wc
      1       6      49
```

We see that most processes are running as SCHED_OTHER on my system.
Some are running as SCHED_FIFO.
And only one is using SCHED_IDLE, which is the lowest priority policy.

5 Alternative to CFS : Other operating systems

5.1 Windows

In the current version of windows multilevel level feedback queue are used. Each of theses are scheduled with round robin.

5.2 IOS

Just a brief description

5.3 MACOS

Just a brief description

5.4 Android

Probably identical to the Linux one

6 Reference and source used

[1]: https://en.wikipedia.org/wiki/O(1)_scheduler
[2]: J. Jose, O. Sujisha, M. Gilesh and T. Bindima, "On the Fairness of Linux O(1) Scheduler," *2014 5th International Conference on Intelligent Systems, Modelling and Simulation*, Langkawi, 2014, pp. 668-674. doi: 10.1109/ISMS.2014.120
URL: https://ieeexplore.ieee.org/document/7280991
[3]: Stallings, W. (2018). *Operating systems: internals and design principles*. Harlow, Essex: Pearson.
[4]: Jones, T. (2006, June 30). Inside the Linux scheduler. Retrieved November 10, 2019, from https://www.ibm.com/developerworks/library/l-scheduler/index.html.
[5]: Jones, . (2009, December 15). Inside the Linux 2.6 Completely Fair Scheduler. November 10, 2019,
Retrieved from https://developer.ibm.com/tutorials/l-completely-fair-scheduler/.
[6]: https://en.wikipedia.org/wiki/Completely_Fair_Scheduler
[7]: Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts*. Hoboken, NJ: Wiley.
[8]: Operating System #22 Completely Fair Scheduling (CFS).
Retrieved from https://www.youtube.com/watch?v=scfDOof9pww.
[9]: Retrieved from https://www.eit.lth.se/fileadmin/eit/courses/eitf60/Rapporter/
Ludwig_Hellgren_Winblad_4547_assignsubmission_file_ludwig.hellgren.winblad.pdf
[10]: https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt
[11]: Priority customisation in Linux. Retrieved from https://en.wikipedia.org/wiki/Nice_(Unix)
and https://askubuntu.com/questions/656771/process-niceness-vs-priority

[12]: Wong, C., Tan, I., Kumari, R., Lam, J., & Fun, W. (2008). Fairness and interactive performance of O(1) and CFS Linux kernel schedulers. *2008 International Symposium on Information Technology*. doi: 10.1109/itsim.2008.4631872
[13]: https://unix.stackexchange.com/questions/407497/how-to-find-scheduling-policy-and-active-processes-priority
[14]: http://man7.org/linux/man-pages/man7/sched.7.html

6.1 Priority customisation in Linux

https://en.wikipedia.org/wiki/Nice_(Unix)
https://askubuntu.com/questions/656771/process-niceness-vs-priority

6.2 Source code of schedulers

Current linux version : CA10000 lines of code
https://elixir.bootlin.com/linux/latest/source/kernel/sched/fair.c

First version introduced : 1200 lines of code
https://elixir.bootlin.com/linux/v2.6.23/source/kernel/sched_fair.c

O(1) scheduler : 7000 lines of code
https://elixir.bootlin.com/linux/v2.6.22.9/source/kernel/sched.c

6.3 Documentation of scheduler

https://www.kernel.org/doc/html/latest/scheduler/index.html

6.4 Explanation of MACOS scheduler

https://developer.apple.com/library/archive/documentation/Darwin/Conceptual/KernelProgramming/scheduler/scheduler.html

6.5 Example of fair scheduling

https://www.youtube.com/watch?v=scfDOof9pww

6.6 vRuntime t increase, I/O bound vs CPU bound processes

https://www.youtube.com/watch?v=scfDOof9pww

https://www.youtube.com/watch?v=scfDOof9pww

6.7 group scheduling

http://man7.org/linux/man-pages/man7/sched.7.html
http://www.informit.com/articles/article.aspx?p=397655&seqNum=6

7.8 other sources:
http://www.informit.com/articles/printerfriendly/101760

https://trepo.tuni.fi/bitstream/handle/10024/96864/GRADU-1428493916.pdf

https://developer.ibm.com/tutorials/l-completely-fair-scheduler/#fig2

MAYBE WILL WORK ON THIS
Alternativ data structure that can be used for processor schedulers:
B-Trees, Suffix Trees, Suffix Arrays, FM-Index
By representing a process via character we can build a sequence of characters, and we can apply Suffix Trees, Suffix Arrays, and FM-Index to manage processes.