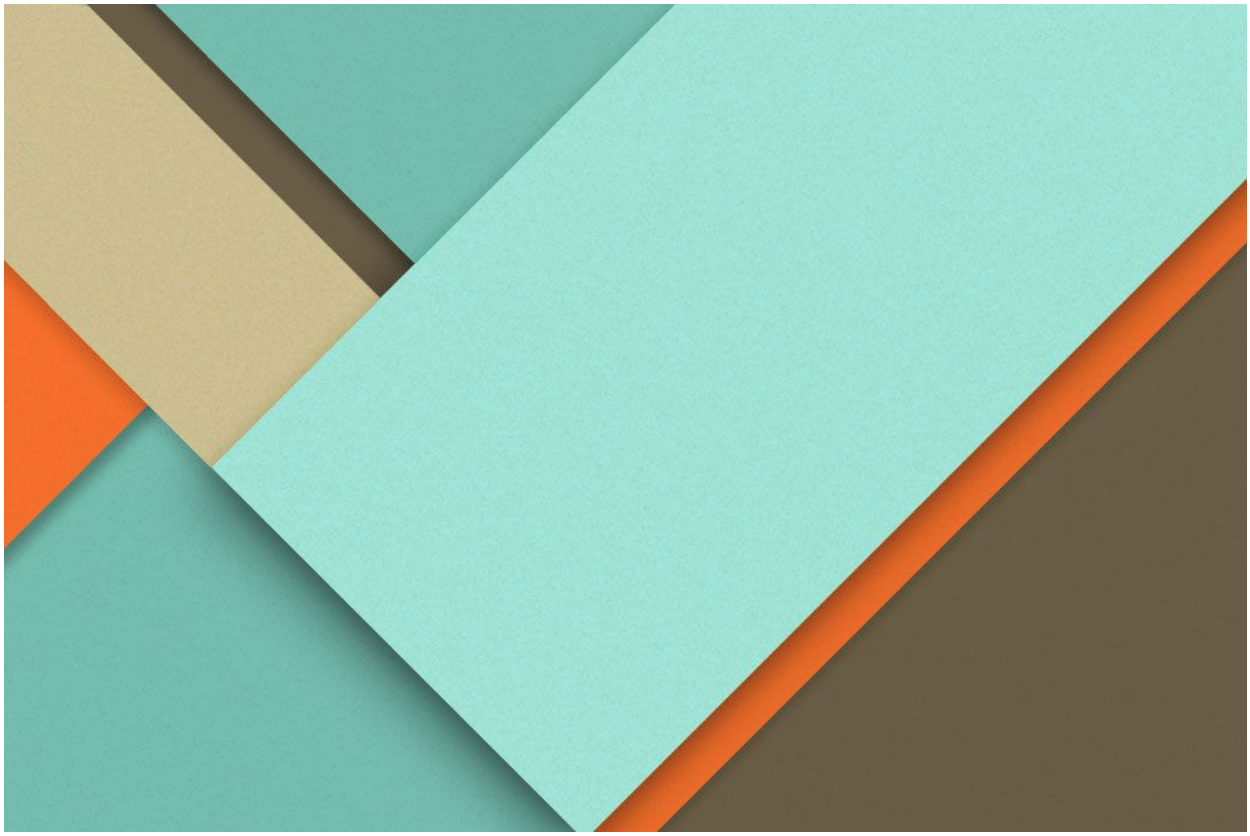


Background Process Scheduling

06.02.2019

B Sasi Kiran Reddy - 17114021
David Gokimmung - 17114023
Gandhi Ronnie - 17114029
Goddu Vishal - 17114034
Suresh Babu - 17114030
Harshavardhan - 17114047
Krishna Yeswanth - 17114055
Saurabh Gautam - 17114069

Overview

The Linux scheduler (pre kernel 3.0) contains 3 built-in scheduling strategies: **SHED_FIFO**, **SCHED_RR** and **SCHED_OTHER**. The **SHED_FIFO** and **SCHED_RR** schedulers are primarily used for real-time scheduling, where a process has time deadlines and requires some guarantee on how soon it will be dispatched. The **SCHED_OTHER** policy, the default, uses a conventional time-slicing method (similar to *round-robin*) to put an upper bound on the amount of time that a process will use the CPU. For **SCHED_OTHER**, a dynamic priority is computed based on a fixed priority and the amount of time a process has been waiting for the CPU to become available. The counter and priority fields in the process control block are the key components in determining the task's dynamic priority. The dynamic priority is adjusted on each timer interrupt.

SCHED_BACKGROUND

We add a new scheduling policy called **SCHED_BACKGROUND** that is designed to support processes that only need to run when the system has nothing else to do. This "background" scheduling policy should run processes when there are no processes in the **SCHED_OTHER**, **SCHED_RR** or **SCHED_FIFO** classes/ queues to run. When there are more than one **SCHED_BACKGROUND** processes ready to run, they should compete for the CPU just like **SCHED_OTHER** processes.

Proposal

1. We have implemented the new policy in **Linux-2.6.24 kernel** in **Ubuntu 8.04** (using **VirtualBox**) since there is no sched.c file in later versions of the linux kernel (3+).
2. The major changes are made in the files /usr/src/linux-2.6.24/kernel/**sched.c** and /usr/src/linux-2.6.24/include/linux/**sched.h**. These are responsible for the functionality of our new scheduler.
3. The **chrt.c** file also has to be modified, converted into a 32/64 bit ELF executable file (depending on the machine), and put in /usr/bin/
4. We take the SCHED_IDLE policy given in linux-2.6.24 kernel as a reference and work upon it to build our new policy.
5. The functionality of both policies is similar except that the priority of SCHED_BACKGROUND needs to be the lowest
6. The new policy is defined in sched.h and required changes are made in chrt.c and sched.c.

7. Our end goal is to design SCHED_BACKGROUND to support processes that only take CPU time when there are no processes of other scheduling classes.

Steps with Terminal code:

1. To start, we need to figure out what version of the kernel we are currently running.

We'll use the uname command for that

$ uname -r

2.6.24-26-generic

2. Since 8.04 is an older, unsupported version of Ubuntu, we have to change the download server for 'apt' command from `archive.ubuntu.com` and `security.ubuntu.com` to `old-releases.ubuntu.com` . For this we used:

$ sudo sed -i -re 's/([a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list

3. Now update packages and add *libc6-dev* package (to support c headers)

$ sudo apt-get update

$ sudo apt-get install libc6-dev

4. Now we need to Install the Linux source for your kernel, you can substitute the kernel number for whatever you are running. We also need to install the curses library and some other tools to help us compile

$ sudo apt-get install linux-source-2.6.24 kernel-package libncurses5-dev fakeroot

5. If you are curious where the Linux source gets installed to, you can use the dpkg command to tell you the files within a package

$ dpkg -L linux-source-2.6.24

6. To make things easier, we'll put ourselves in root mode by using sudo to open a new shell.

$ sudo /bin/bash or

$ sudo su

7. Now change directory into the source location so that we can install. Note that you may need to install the bunzip utility if it's not installed

$ cd /usr/src

$ bunzip2 linux-source-2.6.24.tar.bz2

$ tar xvf linux-source-2.6.24.tar

8. You can also get linux-2.6.24 from https://mirrors.edge.kernel.org/pub/linux/kernel/v2.6/, then extract the file and move the linux-2.6.24 directory to /usr/src

(The file name in the code depends on whether you used step 4 or step 8 to download the kernel source)

$ gunzip linux-2.6.24.tar.gz

$ tar xvf linux-2.6.24.tar (or just right click -> extract)

9. Generate a symlink to linux-2.6.24 with the name linux

$ mv linux-2.6.24 /usr/src

$ ln -s linux-source-2.6.24 linux

10. Make a copy of your existing kernel configuration to use for the custom compile process (This will take care of the Y/n questions that'll be asked during the build)

$ cp /boot/config-`uname -r` /usr/src/linux/.config

11. First we'll do a make clean, just to make sure everything is ready for the Compilation

$ cd /usr/src/linux

$ make-kpkg clean

12. Next we'll actually compile the kernel. This will take a really long time, (took us around 25-30 minutes).

$ fakeroot make-kpkg --initrd --append-to-version=-custom kernel_image kernel_headers

13. This process will create two .deb files in /usr/src that contain the kernel.

14. Please note that when you run these next commands, this will set the new kernel as the new default kernel. This could break things! If your machine doesn't boot, you can hit Esc at the GRUB loading menu, and select your old kernel. You can then disable the kernel in /boot/grub/menu.lst or try and compile again

$ dpkg -i linux-image-2.6.24-custom_2.6.24-custom-10.00.Custom_i386.deb

$ dpkg -i linux-headers-2.6.24-custom_2.6.24-custom-10.00.Custom_i386.deb

15. Now reboot your machine. If everything works, you should be running your new custom kernel. You can check this by using uname. Note that the exact number will be different on your machine

$ uname -r

2.6.24-custom

16. To change the policy of a process to SCHED_BACKGROUND policy (while it's running), we used to command:

$ chrt -x -p 0 <PID> ('$$' substitutes <PID> in a bash (.sh) script)

17. To change the niceness of a process during its runtime, we used

$ renice <niceness> <PID>

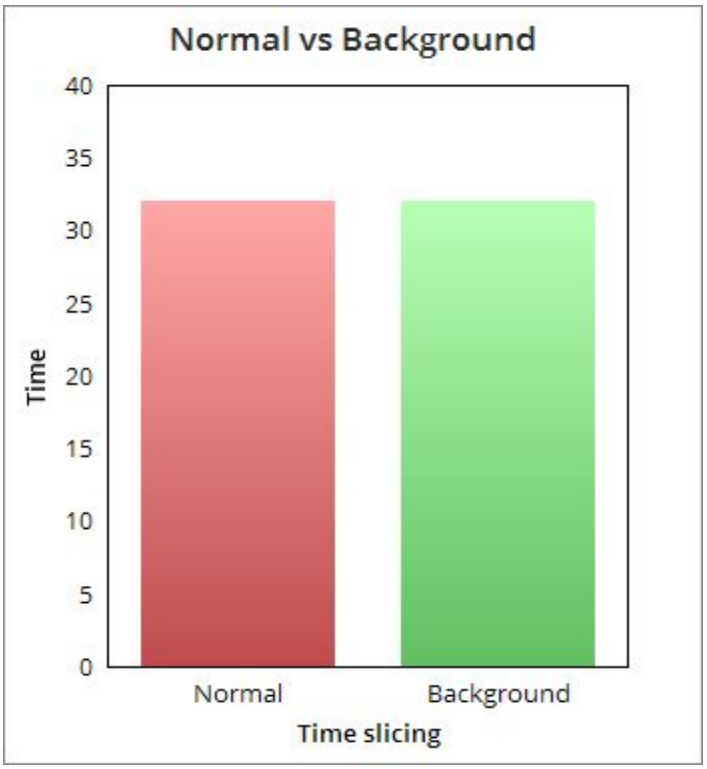
Evaluation :

In all cases, Time is turnaround time. Each counter.sh process takes around 15-16 seconds ***SCHED_BACKGROUND*** *was made by referring to* ***SCHED_BATCH*** *which is a type of* ***SCHED_IDLE*** *policy. Hence, it follows* ***time-slicing*** *as well.* ***'Normal'*** *refers to* ***SCHED_NORMAL*** *and* ***'background'*** *refers to* ***SCHED_BACKGROUND***

1 & 2. Running our counter.sh as a normal process vs running it as a background process. Since in both the cases, counter is the only process in the queue, it executes to its full extent.

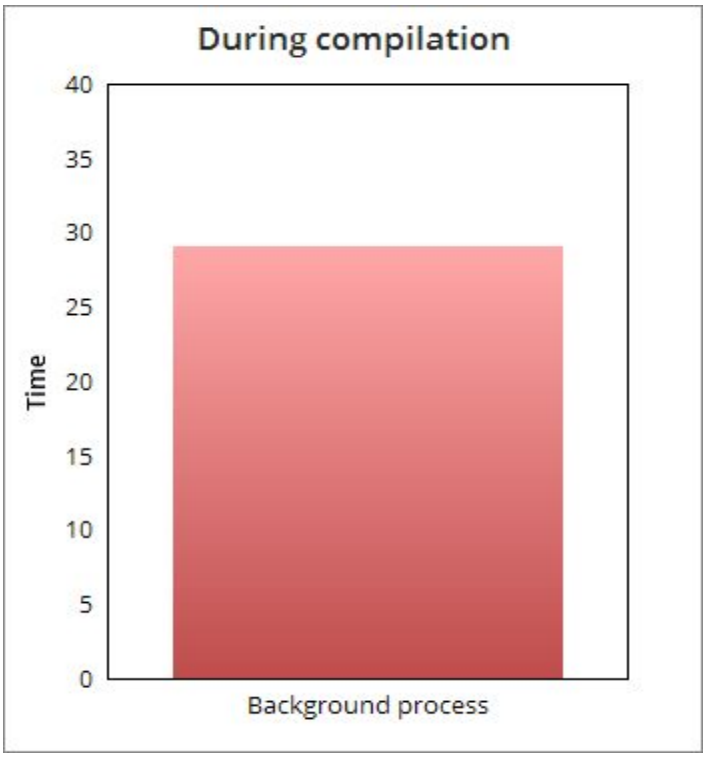
3. Running our counter.sh simultaneously with another counter, both as normal processes. Since *normal* follows time-slicing by default, both finish executing at around twice the individual turnaround time.

4. Running our counter as a background process, simultaneously with another counter.sh running as a normal processes.



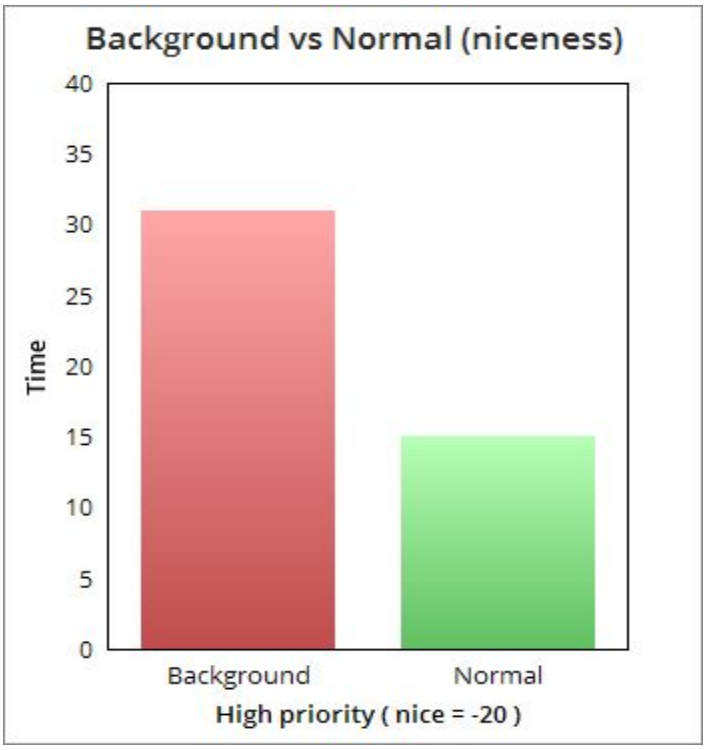
Since, both normal and background were initialized with the same priority values (0), they take CPU time-slices alternatively. (We play with niceness values in later examples).

5. Running our counter.sh as a background process, simultaneously while compiling the kernel (Step 12 in 'Steps with Terminal code') . Background process took twice the individual turnaround time probably because the process of kernel compilation follows normal policy.

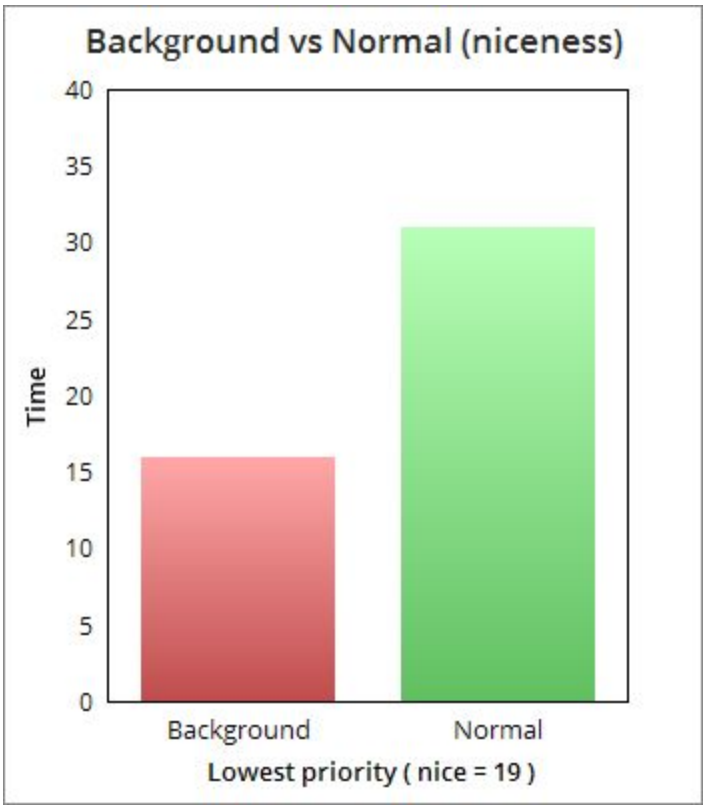


6. Running our counter.sh as a background process, simultaneously with another normal counter.sh that is using nice at the highest priority (nice value -20). Since the normal process is at a higher priority, it executes completely first, i.e., the

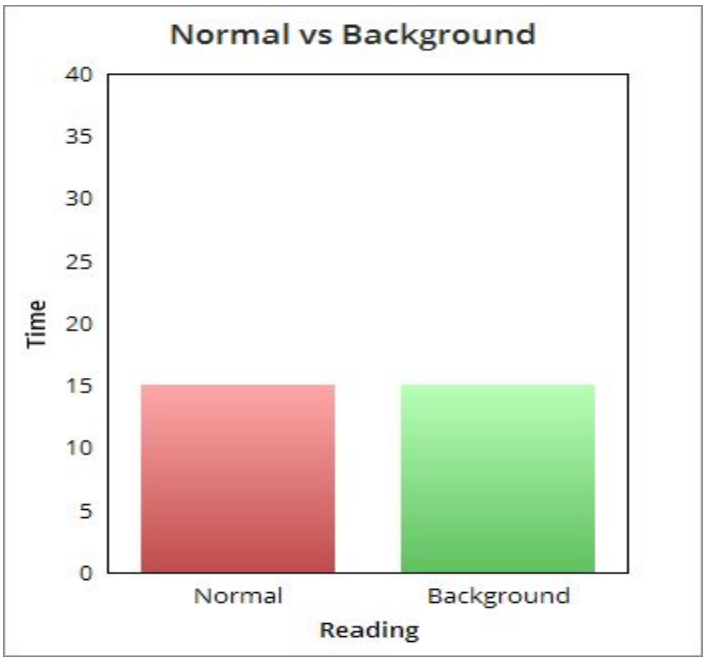
background process only starts executing when the process of higher priority is done.



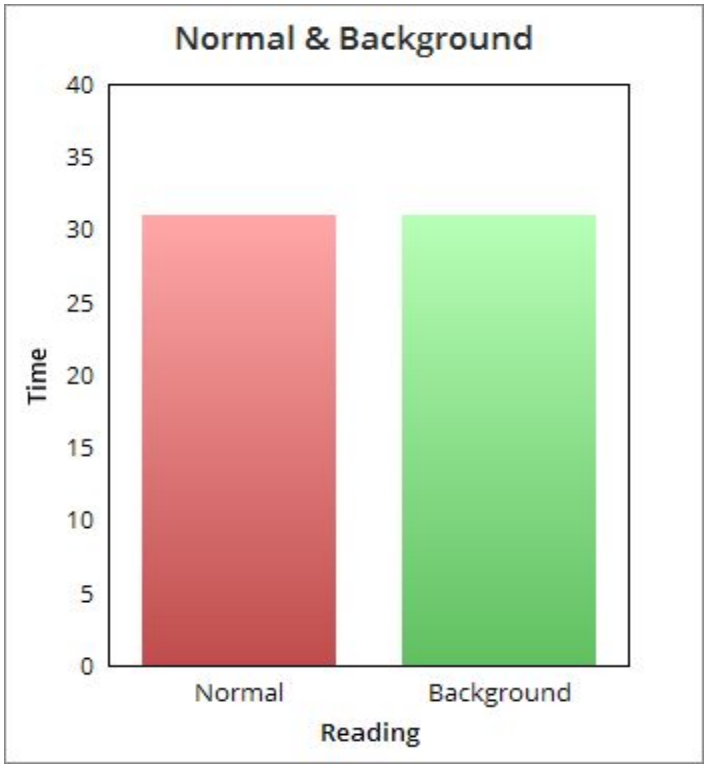
7. Running our counter as a background process, simultaneously with another normal counter that is using nice at the highest priority (nice value 19)



8 (1 & 2). Step 1&2 but instead of using counter.sh, we use reader.sh. reader.sh reads a large file (over 420,000 lines of length 64 characters each - total file size over 50 MB). The reader file size was adjusted to take roughly 15 seconds.



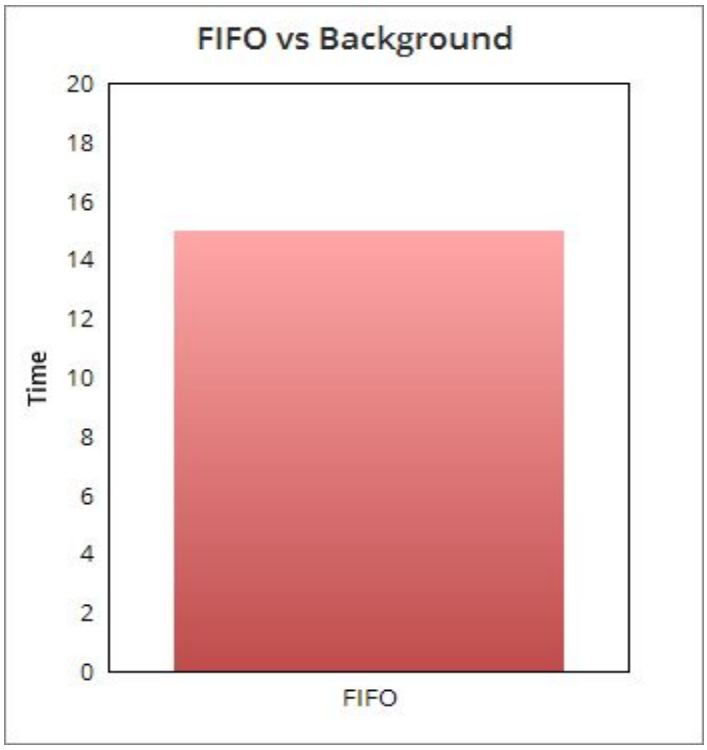
8 (3). Step 3 but instead of using counter.sh, we use reader.sh. We obtain similar results.

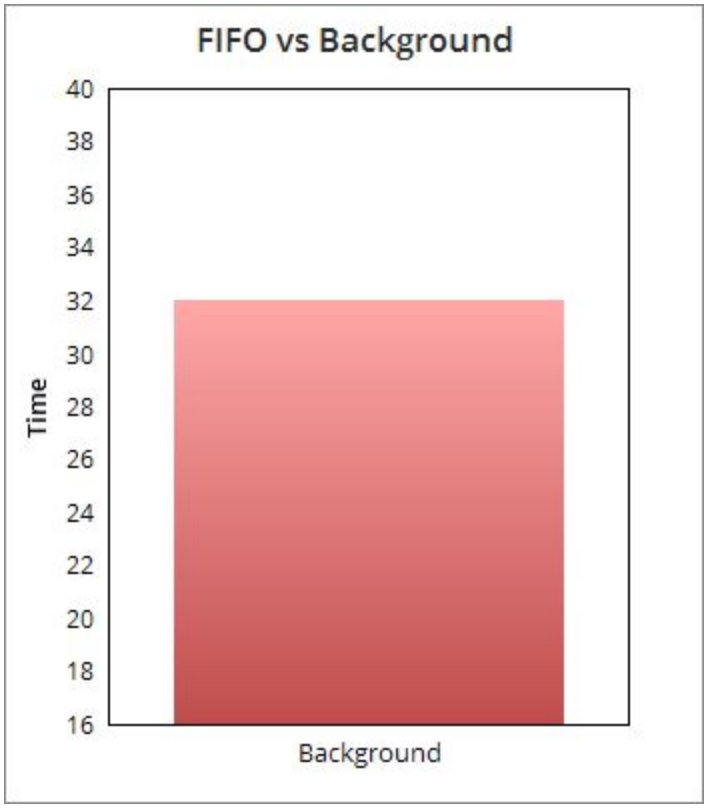


Some custom evaluations we've added: *(Result charts shown in the next page)*

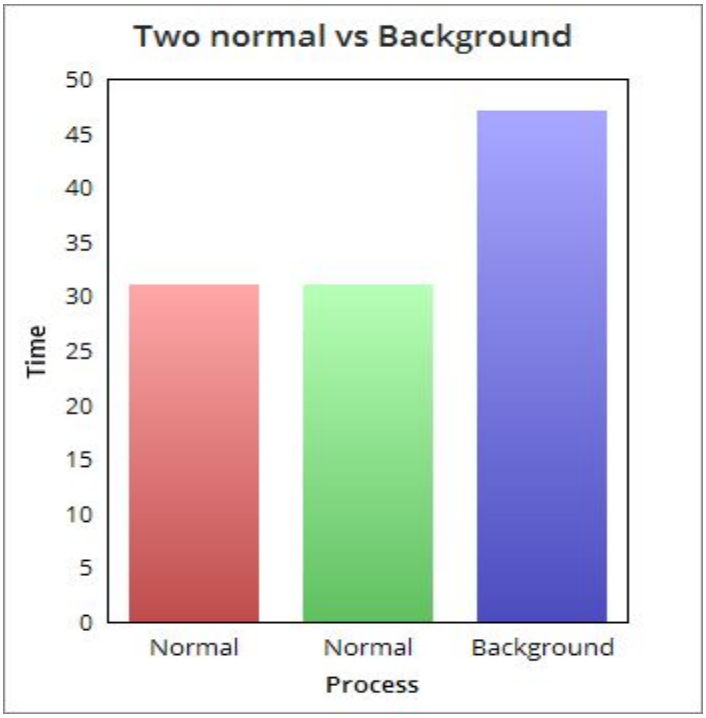
1. **Background counter vs FIFO counter. As expected, the FIFO executes first (from 0 to 15) and background counter executes after it (16-31).**
2. **Running a background counter and 2 normal counters at the same time. The two normals run first alternatively and finish by 30 seconds. The background process starts at 30 and executes till 45.**
3. **Running a background counter and 2 FIFO counters at the same time. As expected, the FIFO counters execute first, one after the other and the background process starts after them.**

1.

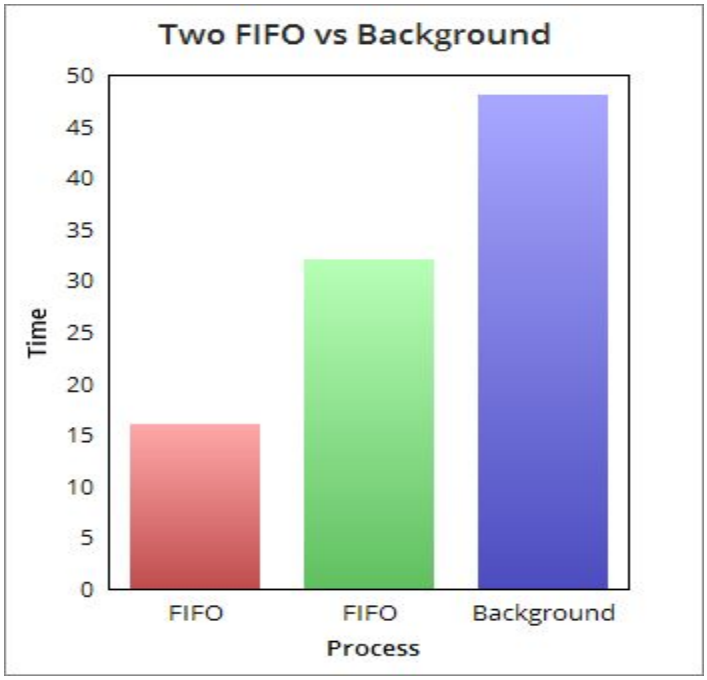

FIFO vs Background
Time
20
18
16
14
12
10
8
6
4
2
0
FIFO


FIFO vs Background
Time
40
38
36
34
32
30
28
26
24
22
20
18
16
Background

2.


Two normal vs Background
Time
50
45
40
35
30
25
20
15
10
5
0
Normal
Normal
Background
Process

3.


Two FIFO vs Background
Time
50
45
40
35
30
25
20
15
10
5
0
FIFO
FIFO
Background
Process

References:

- https://web.cs.wpi.edu/~claypool/courses/3013-A05/projects/proj1/
- https://mirrors.edge.kernel.org/pub/linux/kernel/v2.6/
- https://www.howtogeek.com/howto/ubuntu/how-to-customize-your-ubuntu-kernel/
- https://askubuntu.com/questions/91815/how-to-install-software-or-upgrade-from-an-old-unsupported-release
- https://askubuntu.com/questions/48708/change-niceness-priority-of-a-running-process
- https://lwn.net/Articles/3866/
- https://www.cs.montana.edu/~chandrima.sarkar/AdvancedOS/CSCI560_Proj_main/index.html
- https://www.embedded.com/design/operating-systems/4204929/Real-Time-Linux-Scheduling-Part-1
- https://www.cs.montana.edu/courses/spring2009/518/pages_resources/studentProjects/juanBanda/finalPresentation/Adding%20a%20Scheduling%20Policy%20to%20the%20Linux%20Kernel.ppt