OPERATING SYSTEMS

Group Members:

Bharat [15114018] **email -** bharatraj974@gmail.com

Purushottam Abhisheikh [15114052] **email -** purushottamabhisheikh@gmail.com

Rahul Ram [15114054] **email -** ramcrahul@gmail.com

Ravi Kumar [15114055] **email -** rkshaan35@gmail.com

Shivam Jindal [15118079] **email -** shivamjindal39@gmail.com

PROJECT REPORT

Implementing a new Scheduling Policy for Linux

Introduction:

The Linux scheduler 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.

We have to 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 only runs processes when there are no processes in the SCHED_OTHER, SCHED_RR or SCHED_FIFO classes to run. When there is more than one SCHED_BACKGROUND process ready to run, they should compete for the CPU as do SCHED_OTHER processes.

After completing the above steps we need to make a CPU intensive process and perform this process with each of the available scheduling policies to make a time analysis table.

Approach:

- The project will consist in adding 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.
- We will be working on **linux-2.6.24** which is the default kernel version in **ubuntu-8.04** as for higher versions the encapsulation is increased and due to which the complexity of scheduler code is increased. In higher version there is no sched.c file but a sched directory which is consist of many different parts old scheduler.
- The primary changes are required in **sched.c** and **sched.h** files which are responsible for handling scheduler function in kernel.
- We also need to modify an auxiliary file **chrt.c** which handles all the system calls to kernel-scheduler.
- So the primarily modified files are :
    - */include/linux/sched.h*
    - */kernel/sched.c*
    - */usr/bin/chrt*
- In linux-2.6.24 there is a policy named SCHED_IDLE which already handles background task.
- We will implement a new policy SCHED_BACKGROUND to handle background tasks with lower priority than SCHED_IDLE. This will handle tasks only if queues of other policies are empty.

- We define this new policy in sched.h and make required changes in sched.c. To support making system calls from terminal, we also add this policy in chrt.c, generate its object file and replace it with the default system file.
- There are mainly two scheduling classes in linux-2.6.24 :
    - ***sched_rt.c*** *(for real time processes)*
    - ***sched_fair.c*** *(for fair scheduling)*
- The SCHED_BACKGROUND policy need to handle processes in its queue similar to SCHED_OTHER which follows fair scheduling so it also follows fair scheduling. Thus its operating class is sched_fair.c.

Project steps -

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-generic

2.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

3.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-2.6.24

4.To make things easier, we'll put ourselves in root mode by using sudo to open a new shell. There's other ways to do this, but I prefer this way

$ sudo /bin/bash

5.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-2.6.24.tar.bz2

$ tar xvf linux-2.6.24.tar

$ ln -s linux-2.6.24 linux

6.Make the required changes to implement the new background scheduler in the sched.c , sched.h files and replace them in their original directories.Also change the chrt.c accordingly , compile it and paste the object file chrt.o in user/bin.

7.Make a copy of your existing kernel configuration to use for the custom compile process

$ cp /boot/config- 2.6.24-generic /usr/src/linux/.config

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

$ make-kpkg clean

9.Next we'll actually compile the kernel. This will take a “LONG TIME” maybe 40-50 mins.

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

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

11. 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

12. 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

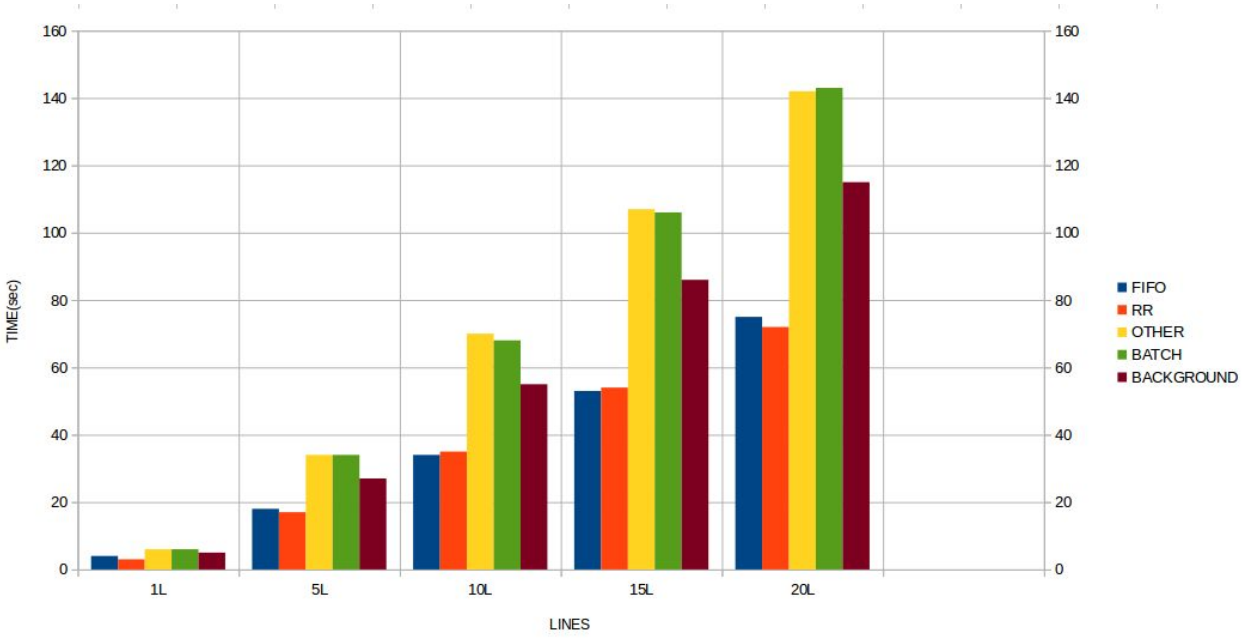
2.6.24-custom

13. To change the policy of a process to **SCHED_BACKGROUND** policy in its runtime, command used :-

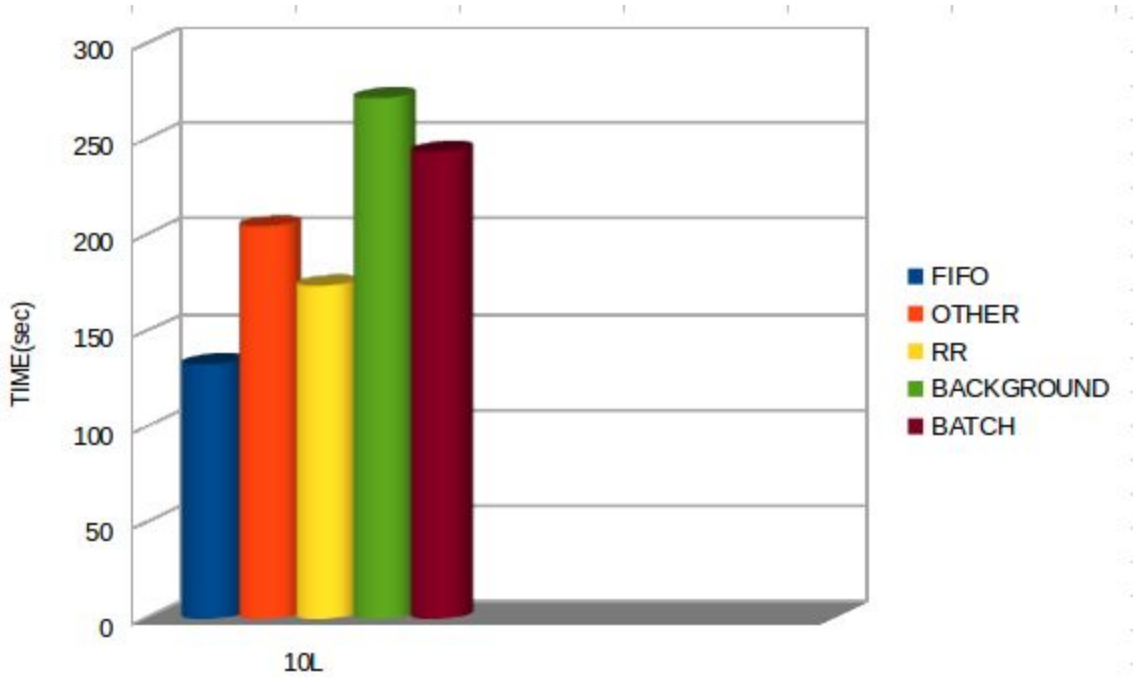
$ chrt -g -p 0 <pid>

Testing with cpu intensive process :

The example test case which we have taken is an loop executing for large number of times :



(single process executing various number of time with large input values)



(for 5 processes running simultaneously)

Note:- All the **modifications** are in the ***sched.c*** and ***sched.h*** files are pre commented by **/*SCHED_BACKGROUND head*/.**

References :

- https://www.ibm.com/developerworks/library/l-completely-fair-scheduler/
- http://web.cs.wpi.edu/~claypool/courses/3013-A05/projects/proj1/
- https://tampub.uta.fi/bitstream/handle/10024/96864/GRADU-1428493916.pdf
- https://help.ubuntu.com/community/Kernel/Compile
- https://www.kernel.org/pub/linux/kernel/v2.6/
- https://www.howtogeek.com/howto/ubuntu/how-to-customize-your-ubuntu-kernel/
- https://launchpad.net/ubuntu/hardy/i386/kernel-package/11.001
- https://www.suse.com/documentation/sles11/book_sle_tuning/data/sec_tuning_taskscheduler_cfs.html?view=print