CSE231 – Operating Systems
Assignment 3
Writeup

Name: Ansh Arora
Roll No: 2019022

Task: Modifying a CFS Scheduler by adding soft rel-time requirement functionalities to it which have a higher priority than vruntime, and then add a system call to modify process's real time requirements.

Description of Code and Implementation –

In the question, we were asked to add a soft real time requirement to the process's task struct's schedule entity (rt_val). This guarantees the task at least x units of time slice. Additionally, any process that has a real time rt_val value given to it, that process will have priority over the process which doesn't have said rt_val guarantees.

Here are the changes that are made to the kernel –

1) The rt_val identifier is added to the sched entity of task struct of a process. This is done in the sched.h file where rt_val is added in the u64 format.

2) The value of rt_val is initialized to zero in the function __sched_fork of core.c file, where other data members of the schedule entity are also initialzed.

3) In the fair.c file of the CFS Scheduler, we have made changes to the entity_before() function, which

is basically a comparator that returns value 1 or 0 on basis of whether the a process needs to come on left on the RB tree or the b process, respectively. While this is traditionally done using the vruntime values, here we add functionality to check the soft real time requirements as well. For this, we add conditional statements. If both the processes a and b are having a soft time requirement, then the process with the lower soft realtime requirement is given the higher priority. If either one of the two processes is having a real time requirement, the process having the realtime requirement is given priority and is made to appear on the left side on the RB Tree. In case both a and b don't have soft guarantees, then the normal scheduling placement using vruntime is followed.

4) Another change in fair.c file is made in the function update_curr(). Here, we subtract the execution time of the process from the x slices of guaranteed time (rt_val, our soft realtime requirement) after it runs. The delta_exec value is decremented from the rt_val value if and only if the process has a rt_val value, for which a check has been added in the update_curr() function. If that isnt the case, then the normal functioning occurs where the vruntime value is decremented as it is.

The rtnice system call is added using the SYSCALL_DEFINE2 function, which takes in the process in which soft realtime guarantee is to be added and the soft real time guarantee value as its parameters. The syscall uses the find_get_pid and pid_task methods to get the task_struct of the given pid process, and then allots it the passed real time values. Errors are handled. The successful processing of this system call is also accompanied with a kernel

message, which can be accessed from the terminal using the dmesg command.

The syscall is implemented by adding the syscall in the syscall table and also adding the folder of its presence in the core-y of the main Makefile.

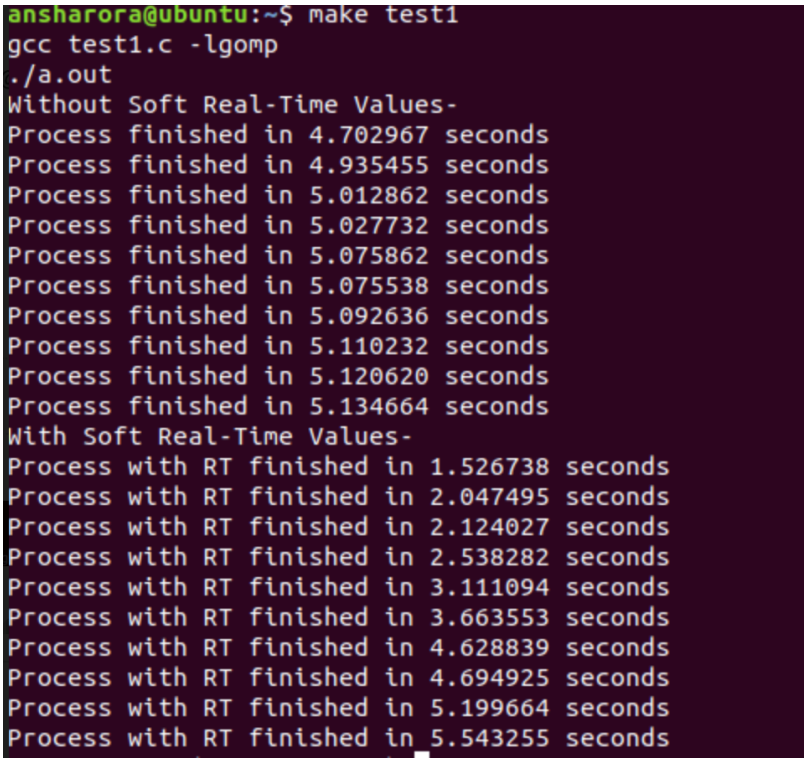
Inputs Taken from User –

The syscall takes in the pid, and soft realtime values in seconds. The value is then multiplied by 10^9 inside the system call by itself, to convert it into nanoseconds. The user can run the two testfiles by the make commands as given in the below Makefile.

```
compile:
    gcc -lgomp test.c

run: compile
    ./a.out

compile2:
    gcc -lgomp test2.c

run2: compile2
    ./a.out

clear:
    rm a.out
```

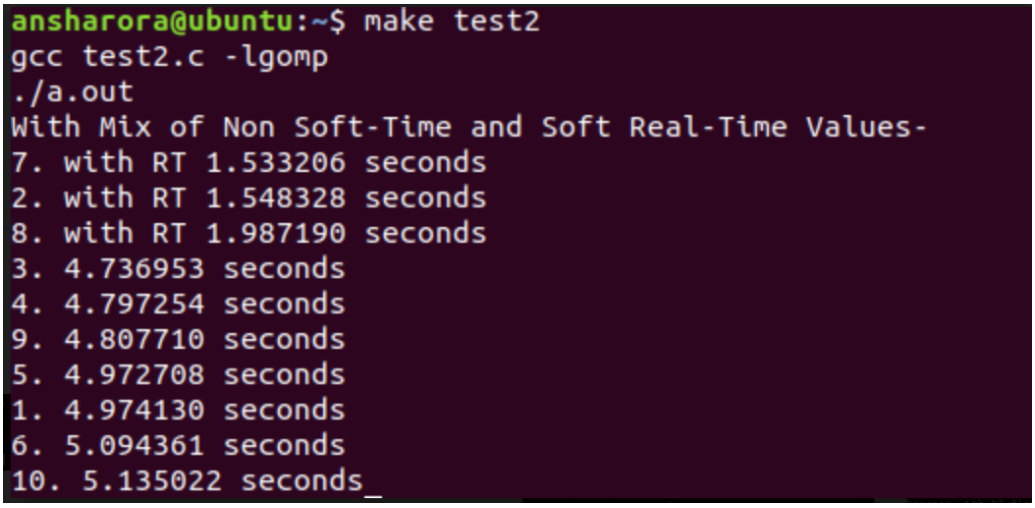
Output Expected -

In the first test file, I have shown the difference between running with and without soft real time guarantees. I create 10 processes for each, individually and make them run concurrently.



As we can see in the attached screenshot, in the case without real time values, the time taken in seconds comes out to be much higher than it does so in the case of soft real time values, where the processes having the real time values are given the highest priority, hence being able to run much faster than in the case when no real time guarantees are passed.

In the second test file, I have run 10 processes, few of them having real time guarantees and the others not, concurrently. The process for which real time guarantees are offered are picked randomly using rand(). All processes run the same function once again.



In this case, as we can see, the randomly selected processes given real time guarantees (depicted using with RT) end up running much faster performance than processes that don't have these soft realtime guarantees. Thus the processes having soft realtime values are given priority over other processes.

All above processes are created as child processes inside the testfiles and exit() on termination. They all run the same function func() in the given time.

Error Values and How to Interpret Them-

1. ESRCH (Errno = 3) No such Process Found. This is generated when the given pid doesn't have a process attached to it. This is handled in the test.c file as an errno generated, and the error number is printed out along with error message using perror() and stderror().

2. EINVAL (Errno = 22) Invalid argument. This is generated when the given pid or soft real time value (syscall's arguments) lie in a range that is unacceptable to the system (when they are negative). These are again handled using perror() and stderror().

Additionally, in my test file I have created a function to check the correct functioning of the errors using error check.