Aryan Behal Assignment-3 2019026

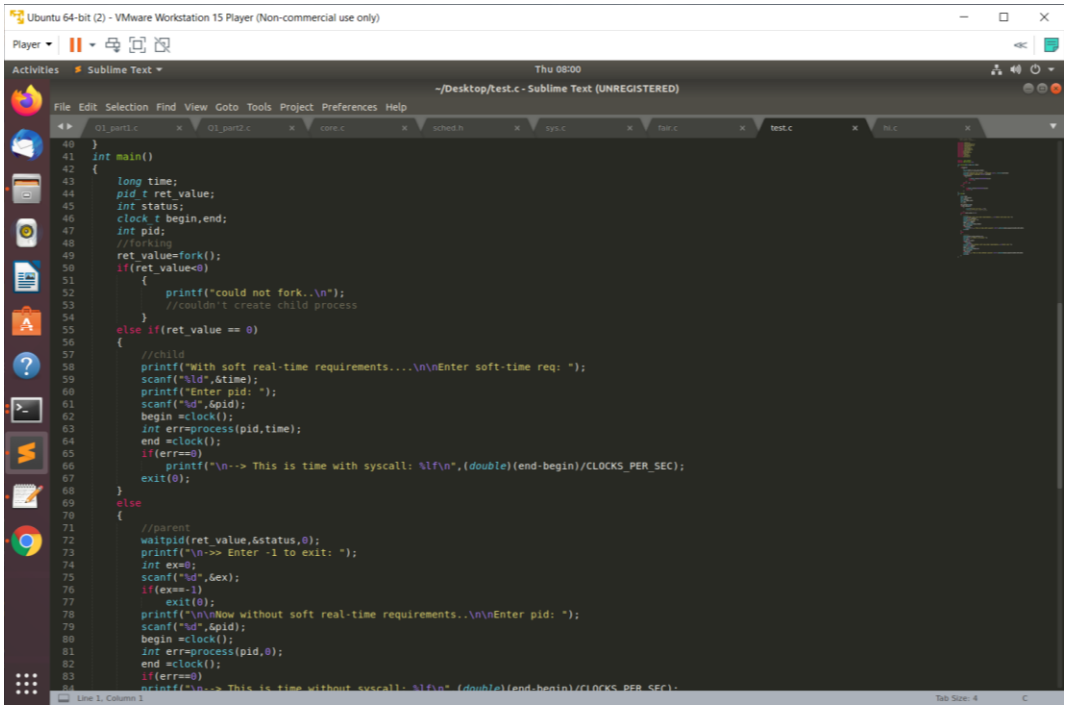
Kernel version: v5.9

How to use:

You just need to traverse to the folder “2019026_OS_Assignment_3” and use “make run” in terminal to run test.c

I am using greater the soft time value passed, greater the priority logic as said by prof. Arani in comments.

My test.c code



Code Description and logic

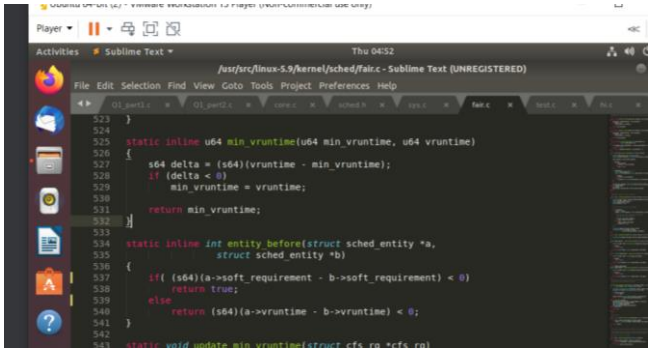
- Test.c
  a. In my test.c code, I am making a parent and child process using **fork ()** and asking user for the pid and soft-time requirements in the child process. Then I am calling calling a function **process ()** where I am calling the **syscall ()** in child’s case to set up the soft real-time requirements.
  b. Also, I am using a large useless for loop to increase execution time to see changes easily if any in process ().
  c. In the child and parent process, I am noting the time for which process () ran when called using **clock ()** that returns the system time (data type **clock_t**). I am taking difference of the time when process () was called and when we returned from it. It is the difference of these noted times that is indicative of our scheduler activity.

- Changes for modifying scheduler

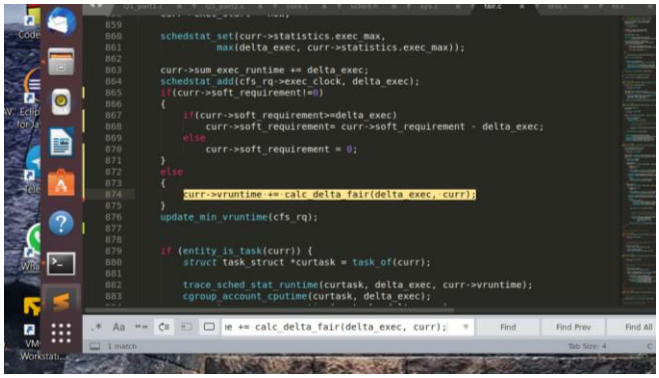
  ➔ All the major scheduling related details for a process like vruntime are handled by a data structure “sched_entity” in linux-5.9**/include/linux/sched.c** file. Since, we needed to use an extra

attribute for determing soft real-time processes, I added a new variable called "soft_requirement" to it and initialised it to 0 (default) in l**inux-5.9/kernel/sched/core.c**.

- ➔ Thereafter, I just needed a logic to give a process with greater soft_requirement more priority. I used a simple hack for this. I didn't change the way CFS scheduler was implemented as a whole. I just changed the way rb_tree was constructed using soft_requirement.
- ➔ I changed function **entity_before()** that served as comparator for making **RB tree** such that the leftmost position of RB tree was given to the process with greatest soft real-time requirement. This way it got processed the earliest (as we know from simple CFS technique).
- ➔ Basically, I ask user for a time interval for which a process must be given priority. The greater the value of this time interval, greater will be the priority for the process. After a process runs for this interval, it becomes a normal process and follows normal CFS scheduling.



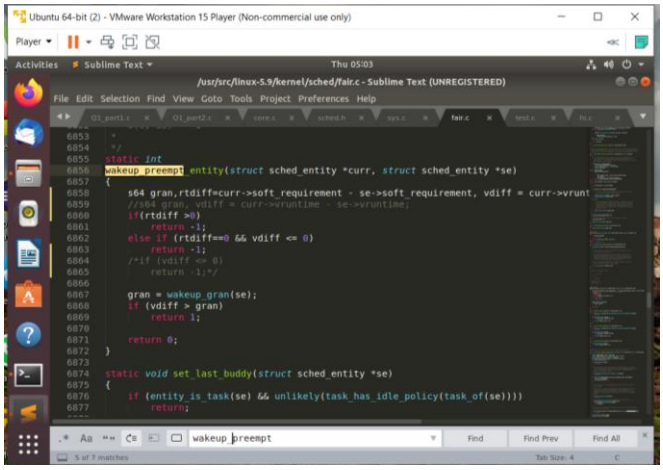
- ➔ Other changes I made are:



This change provides the real time scheduling behaviour.

We basically check soft_requirement till it is greater than 0 and make it the most important prioritised process till it is executed for the defined soft_requirement time. After that it is assigned a proper vruntime as by CFS.

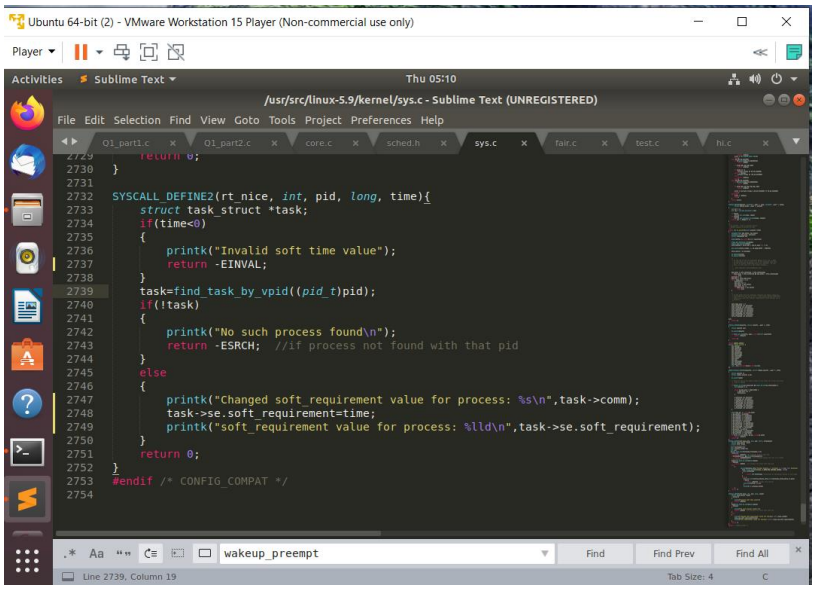
We don't assign a vruntime to process until it is run for enough duration given by soft_requirement. After that it runs like a normal process with normal CFS scheduling.



This checks if some greater priority process has appeared or not and pre-empts accordingly.

These changes ensure that the process with the highest soft_requirement is the leftmost node of the rb_tree and given priority above all lower priority processes till the time it runs for that period.

- <u>Changes for Syscall</u>



  - Basically, I am taking 2 arguments, the pid of process and soft real time requirement value for the process and just changing the soft_requirement to value passed by simple assignment. Syscall is made in linux-5.9/kernel/sys.c
  - I have used task_struct structure for storing process info, function "find_task_by_vpid(pid_t pid)" for finding the process with passed pid and "printk" for printing to kconsole.

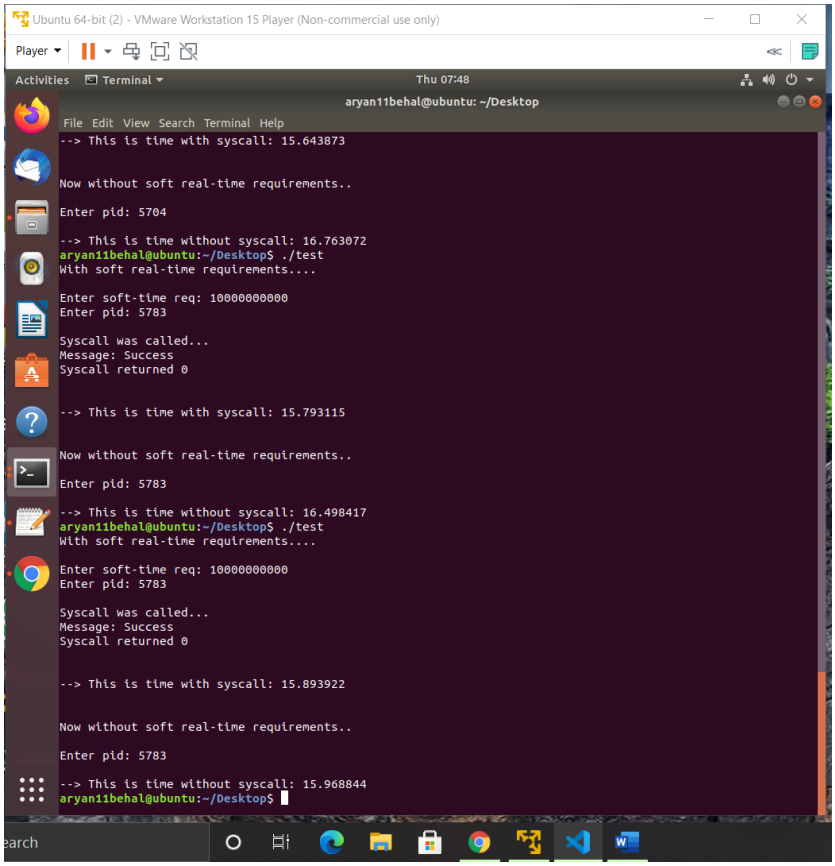
➔ <u>User Input</u>

  - Soft real-time requirement- **long int** (should be a valid long value, not string or char)

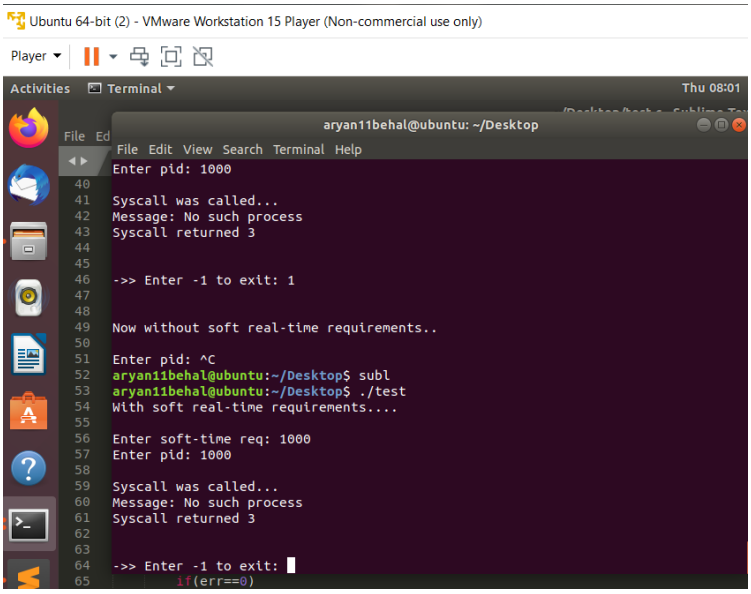
        - It is the fixed interval for which process will be given priority.
        - Generally assign a big value to this variable as it assigns time in nanoseconds.
    - Pid – **pid_t** (valid int value, >0, not string or char)
        - The process id whose soft real time value we wish to change.

➔ <u>Output</u>

- The time taken by process () to run will be displayed when ran it in child by calling syscall () to give higher priority and when no syscall was called in parent.
    - 2 float values will be displayed.
    - A difference in the 2 values will appear. More often than not, the process () called by child will take smaller time because it was given real time property. So, its execution was prioritised by processor and hence executed faster.
    - But process () by parent is just another normal process, so processor takes a bit longer to process it
    - But we might see opposite results but favourable results appear more showing successful implementation of our modified scheduler.



- Error messages or success for syscall() will also be displayed.



- ➔ <u>Error</u>
  - I have handled 2 errors:
    - ESRCH- when process with passed pid not found.
      - Errno = 3
    - EINVAL- when soft_requirement passed is not valid value (<0)
      - Errno = 22
  - Respective error messages will be displayed using strerrno ().
- ➔ Test Cases
  - case 1:
    - pid: 1000 (let us assume no process with pid 1000)
    - soft_real_time_requirement: 10000000000
  - case 2:
    - pid: 5873 (if say 5873 is a valid pid running at that time)
    - soft_real_time_requirement: -50
  - case 3:
    - pid: 5873 (if say 5873 is a valid pid running at that time)
    - soft_real_time_requirement: 10000000000

Reference: 1. https://github.com/shagunuppal/Modifying-CFS-Scheduler

2. https://www3.cs.stonybrook.edu/~dongyoon/cse506-f19/lecture/lec11-scheduler2.pdf

3. https://stackoverflow.com/questions/19181834/what-is-the-concept-of-vruntime-in-cfs#:~:text=The%20vruntime%20is%20the%20virtual,vruntime%20of%20a%20cfs%20runqueue

4. https://oakbytes.wordpress.com/2012/07/03/linux-scheduler-cfs-and-virtual-run-time/