

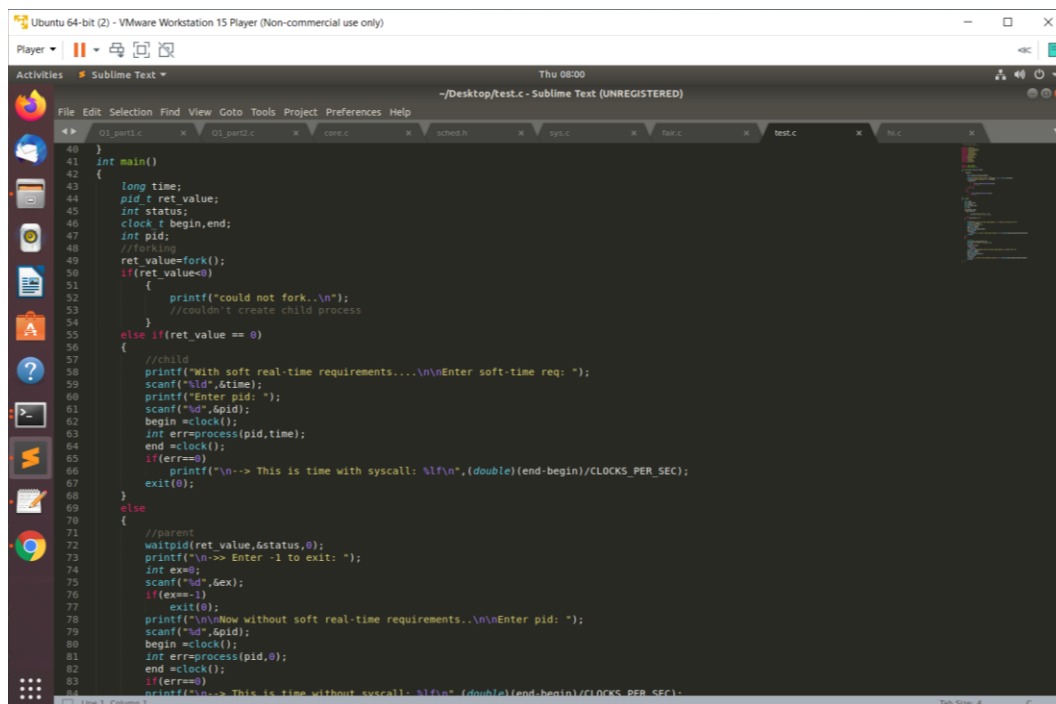
## Kernel version: v5.9

### 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



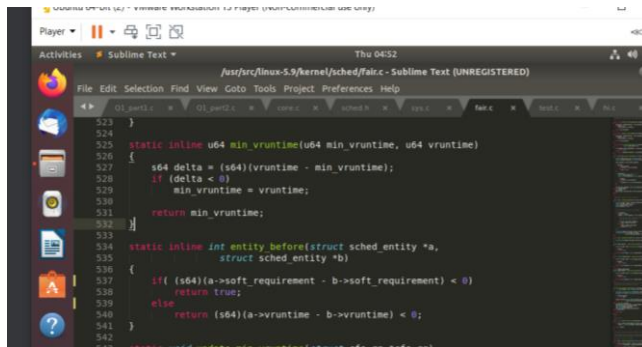
```
1  }
2  int main()
3  {
4      long times;
5      pid_t ret_value;
6      int status;
7      clock_t begin, end;
8      int pid;
9      //forking
10     ret_value=fork();
11     if(ret_value<0)
12     {
13         printf("could not fork.\n");
14         //couldn't create child process
15     }
16     else if(ret_value == 0)
17     {
18         //child
19         printf("With soft real-time requirements...\nEnter soft-time req: ");
20         scanf("%ld",&time);
21         printf("Enter pid: ");
22         scanf("%d",&pid);
23         begin =clock();
24         int err=process(pid,time);
25         end =clock();
26         if(err==0)
27             printf("\n-> This is time with syscall: %lf\n", (double)(end-begin)/CLOCKS_PER_SEC);
28         exit(0);
29     }
30     else
31     {
32         //parent
33         waitpid(ret_value,&status,0);
34         printf("\n-> Enter -1 to exit: ");
35         int ex=0;
36         scanf("%d",&ex);
37         if(ex==-1)
38             exit(0);
39         printf("\n\nNow without soft real-time requirements...\nEnter pid: ");
40         scanf("%d",&pid);
41         begin =clock();
42         int err=process(pid,0);
43         end =clock();
44         if(err==0)
45             printf("\n-> This is time without syscall: %lf\n", (double)(end-begin)/CLOCKS_PER_SEC);
46     }
47 }
```

### 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

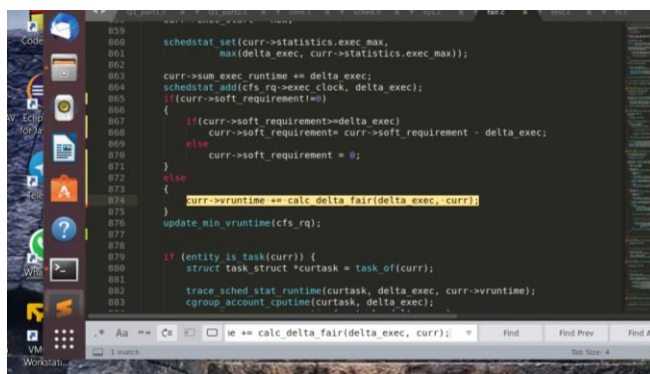
attribute for determining soft real-time processes, I added a new variable called "soft\_requirement" to it and initialised it to 0 (default) in `linux-5.9/kernel/sched/core.c`.

- ➔ 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.



```
523 }
524
525 static inline u64 min_vruntime(u64 min_vruntime, u64 vruntime)
526 {
527     s64 delta = (s64)(vruntime - min_vruntime);
528     if (delta < 0)
529         min_vruntime = vruntime;
530
531     return min_vruntime;
532 }
533
534 static inline int entity_before(struct sched_entity *a,
535                               struct sched_entity *b)
536 {
537     if ((s64)(a->soft_requirement - b->soft_requirement) < 0)
538         return true;
539     else
540         return (s64)(a->vruntime - b->vruntime) < 0;
541 }
542
543 static void update_min_vruntime(struct cfs_rq *cfs_rq)
```

➔ Other changes I made are:

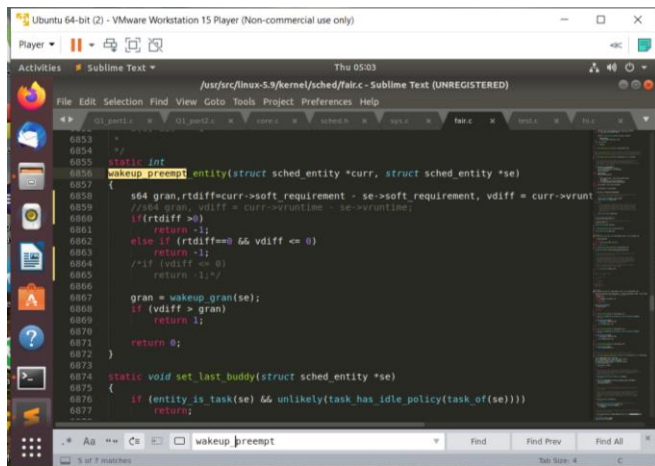


```
059 schedstat_set(curr->statistics.exec_max,
060               max(delta_exec, curr->statistics.exec_max));
061
062 curr->sum_exec_runtime += delta_exec;
063 schedstat_set(cfs_rq->exec_clock, delta_exec);
064 if (curr->soft_requirement != 0)
065 {
066     if (curr->soft_requirement > delta_exec)
067         curr->soft_requirement = curr->soft_requirement - delta_exec;
068     else
069         curr->soft_requirement = 0;
070 }
071
072 else
073 {
074     curr->vruntime += calc_delta_fair(delta_exec, curr);
075 }
076 update_min_vruntime(cfs_rq);
077
078 if (entity_is_task(curr)) {
079     struct task_struct *curtask = task_of(curr);
080
081     trace_sched_stat_runtime(curtask, delta_exec, curr->vruntime);
082     cgroup_account_cputime(curtask, delta_exec);
083 }
```

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.

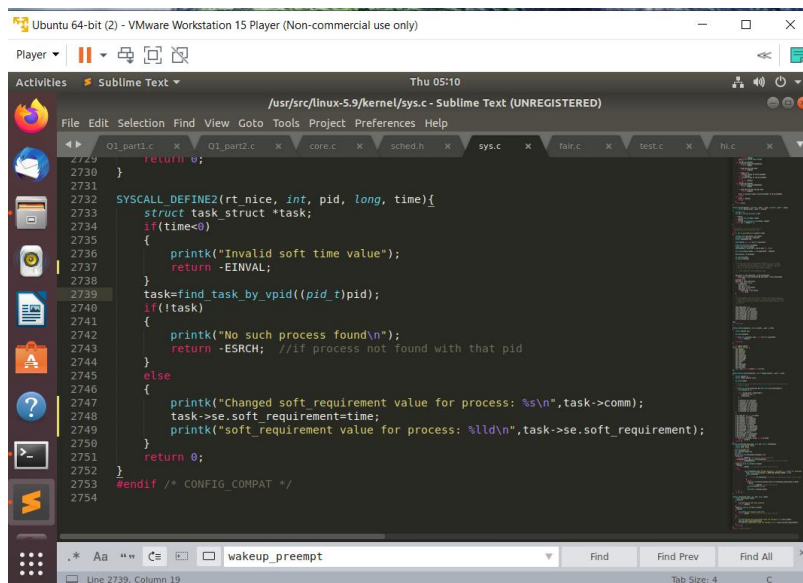
We don't assign a vruntime to process until it is run for enough duration given by soft\_requirement. After that it runs like a normal process with normal CFS scheduling.



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.

- Changes for Syscall



- 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.

➔ User Input

- 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.

## → Output

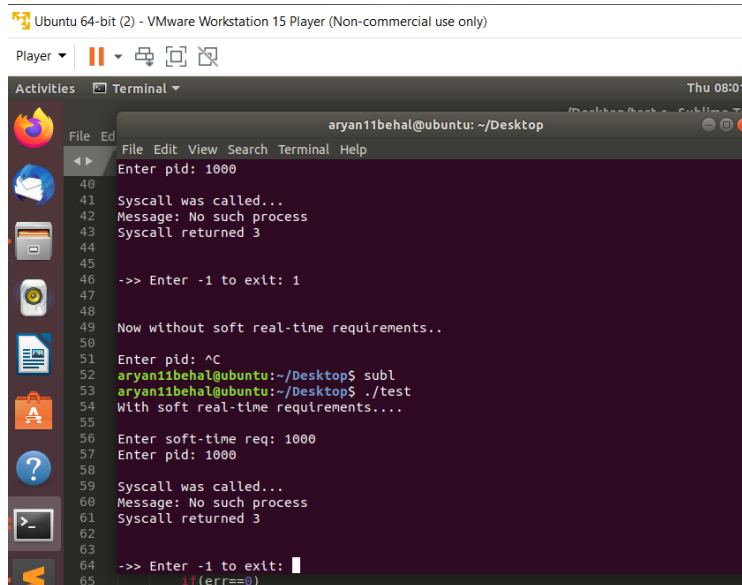
- The time taken by process () to run will be displayed when ran it in child by calling syscall () to give higher priority and when no syscall was called in parent.
  - 2 float values will be displayed.
  - A difference in the 2 values will appear. More often than not, the process () called by child 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.

```

Ubuntu 64-bit (2) - VMware Workstation 15 Player (Non-commercial use only)
Player
Thu 07:48
aryan11behal@ubuntu: ~/Desktop
File Edit View Search Terminal Help
--> This is time with syscall: 15.643873
Now without soft real-time requirements..
Enter pid: 5704
--> This is time without syscall: 16.763072
aryan11behal@ubuntu:~/Desktop$ ./test
With soft real-time requirements....
Enter soft-time req: 10000000000
Enter pid: 5783
Syscall was called...
Message: Success
Syscall returned 0
--> This is time with syscall: 15.793115
Now without soft real-time requirements..
Enter pid: 5783
--> This is time without syscall: 16.498417
aryan11behal@ubuntu:~/Desktop$ ./test
With soft real-time requirements....
Enter soft-time req: 10000000000
Enter pid: 5783
Syscall was called...
Message: Success
Syscall returned 0
--> This is time with syscall: 15.893922
Now without soft real-time requirements..
Enter pid: 5783
--> This is time without syscall: 15.968844
aryan11behal@ubuntu:~/Desktop$

```

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



```
Ubuntu 64-bit (2) - VMware Workstation 15 Player (Non-commercial use only)
Player
Activities Terminal
aryan11behal@ubuntu: ~/Desktop
File Edit View Search Terminal Help
Enter pid: 1000
40
41 Syscall was called...
42 Message: No such process
43 Syscall returned 3
44
45
46 -> Enter -1 to exit: 1
47
48
49 Now without soft real-time requirements..
50
51 Enter pid: ^C
52 aryan11behal@ubuntu:~/Desktop$ subl
53 aryan11behal@ubuntu:~/Desktop$ ./test
54 With soft real-time requirements....
55
56 Enter soft-time req: 1000
57 Enter pid: 1000
58
59 Syscall was called...
60 Message: No such process
61 Syscall returned 3
62
63
64 -> Enter -1 to exit:
65 if(err==0)
```

## → Error

- 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 strerror().

## → Test Cases

- case 1:
  - pid: 1000 (let us assume no process with pid 1000)
  - soft\_real\_time\_requirement: 1000000000
- 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: 1000000000

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/>