CSE231 - Operating Systems
Assignment 3
Exercise 3.1

Name: Abhik S Basu
Roll Number: 2020165
Section: A
Branch: CSE

Task: Modifying vruntime of Linux CFS Scheduler

To approach this question, we need to first understand how exactly the vruntime of a particular process is calculated and where exactly is it calculated. Following that we will establish a sys call:

They are as follows:
sched.h: /home/kern/build/linux-5.14.7/include/linux/sched.h
fair.c: build/linux-5.14.7/kernel/sched/fair.c
core.c: /home/kern/build/linux-5.14.7/kernel/sched/core.c

Here are the changes that we have made to the follwing files:

1) In sched.h we find the sched_entity structure and add an attribute to it which will store the amount of delay that has to be given to the vruntime of the process.

2) In fair.c, changes are made to the update_curr() function as well . In this we add the newly added delay attribute when we are computing the curr_vruntime of the process there we add the delay attribute.

3) In core.c , we initialise all the members of sched_entity to zero. So we just add the delay attribute and set that to zero as well.

Now after making these changes to the kernel we now add the definition of the syscall in the syscall table and further define it in the sys.c document.

In the syscall, we take in two parameters which are the ID of the process whose vruntime has to be increased. And the other parameter is the delay time which is taken by us in millisecond. We convert the millisecond to nanoseconds in out syscall by multiplying it by 1e6. Here we also initialise the structure pid_struct since we will assign it the PID on which the delay time has to be added. After doing error handling on the vruntime and pid we do the multiplication, after that we assign the struct the PID and then using pid_task() function we assign the task_struct the pid_struct. Then we change the vruntime of this process by adding the delay, the delay has been computed and assigned to the newly added attribute. This is basically what are syscall is doing. I am also printing the newly delayed time in order to show it on the system log that how the vruntime has been affected for each of the processes. This will tell us the scale of exponents by which vruntime has been affected.

Following that we execute the make and then we will see the changes.

Now I will explain how the testing works:
We have created two programmes test and test2. In ‘test’ we first compute the time the process runs for and then in ‘test2’ we compute the time the process runs for when the syscall is invoked. We will now notice the difference in the execution time. Since the process is getting changed in the update_curr() so we will see the difference in the execution time.

We can test all of this by running the command as follows:

make

```
all: t1 t2 run

t1: test.c
    gcc test.c -lgomp -o test

t2: test2.c
    gcc test2.c -lgomp -o test2

clean:
    rm test test2

run:
    ./test && ./test2
```

```
[kern@artixcse231 ~]$ make
gcc test.c -lgomp -o test
gcc test2.c -lgomp -o test2
./test && ./test2
Nice value of process is : 5
Timeslice for non Sys Call process is: 0 seconds, 16666665 nanoseconds
Finished executing for non Sys Call in 3.895953 seconds
Nice value of process is : 5
Timeslice for Sys Call process is: 0 seconds, 16666665 nanoseconds
Finished executing for Sys Call in 3.948575 seconds
[kern@artixcse231 ~]$
```

```
[kern@artixcse231 ~]$ make
gcc test.c -lgomp -o test
gcc test2.c -lgomp -o test2
./test && ./test2
Nice value of process is : 5
Timeslice for non Sys Call process is: 0 seconds, 16666665 nanoseconds
Finished executing for non Sys Call in 3.996974 seconds
Nice value of process is : 5
Timeslice for Sys Call process is: 0 seconds, 16666665 nanoseconds
Finished executing for Sys Call in 4.108231 seconds
```

We can check the kernel log, by using the command as follows:

```
sudo dmesg | tail
```

The output for that will be as follows:

```
vruntime of PID: 382 is now 81116401452980713 54
PID Delay : -> 1000000000000000
vruntime of PID: 383 is now 88006471916370144 06
PID Delay : -> 10000000000
vruntime of PID: 383 is now 186471204942
PID Delay : -> 186471306972
```

We can notice that the difference or increase in the vruntime will always be scaled to the amount of digits in the delay. We can compare this to previous runtime as well by doing another printk() in our sys call definition and it will work. The difference b/w the vruntime will be of the form 1e(x-y), x will be digits in bigger delay and y is digits in smaller. And the difference will approximately be equal to 1e(x-y).