HARSH KUMAR AGARWAL
2019423

Download the kernel using wget command.
https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.9.1.tar.xz
I used the link provided above.
Extract it to /usr/src/ now enter the folder.

Files to modify : -------
To modify scheduler:--
We first add a variable(named it rtnice) which will store the soft time value passed by the user, and initialize it to 0.
This variable is of u64 type; it means, it has unsigned 64 bit memory(equivalent to long long int).
u64 rtnice; //inside iclude/linux/sched.h
p->se.rtnice= 0; //inside kernel/sched/core.c
Now we make modifications inside the kernel/sched/fair.c file, here we edit two functions pick_next_entity() and update_curr().

To add system call:--
Here create a new folder with the syscall name we want to create(rtnice). Now make 2 files .c and a Makefile.
Write code in the .c file(explained later). After that edit the original Makefile. Now add the new system call to the system call table.

Explanation of code:-----
Scheduler part:--
We add our own variable(rtnice) to the struct sched_entity and initialize it to 0.

In fair.c we modify our process scheduler,

update_cur(): we update our variable every time it is called. It updates(reduces the value of rtnice) after checking if its value is greater than 0 or not. If it is greater than 0 it means it must have runned for a few times and we need to update its value.

pick_next_entity() : This is the most important part.We create a variable which will store the min value of rtnice among all the process. Here before returning the current process, we go through all the process(using for_each_process) and check if anyone of them have rtnice greater than 0. If so we check if the process is in the running queue, if it is so, then we compare its value with the variable we initialised and save it. Later we return the value.

System call part:--
Include all the necessary headers
I used SYSCALL_DEFINE2 as we needed 2 parameters(pid number and the value of rtnice passed).
Now i go through task_struct and check if the pid number provided is equal if so, we make flag 1 and initialise a variable(rtnice) with the value passed. We use printk to write to the kernel log.
If the soft time value passed is less than zero it will return with an error.
If the given value is not in range of PID then it will return with an error.
If we did not find the pid number provided. The flag will remain 0 and return -1.

On successful execution it will return 0

I found the difference file before compilation
I used: $diff ruN originalfolder updatedfolder > diff.txt
This created the diff file.
To patch this file , extract the tar file and apply patch using
$patch -s -p0 < diff.txt
This will make all the changes to the extracted folder, now move the changed folder to /usr/src/
Here go inside the folder, save config file(using $sudo make menuconfig) and compile(using $sudo make -jn && sudo make modules_install install)
The compilation will take time
After compilation restart, and test the code using the test.c file provided.
Compile and run the test.c file using Makefile

On running the test.c file, it asks the user for the soft time value.
If everything works fine the system call will return 0 and the terminal will show the message. we can see the kernel message using $dmesg
It will print the time taken by 2 processes(child and parent process) one executed by calling the system call and another without it.
There is no other difference between the child and the parent process except the syscall part.

If anything goes wrong it shows a message showing there was an error with appropriate error message.
If it was not able to find the PID number provided it will return 1, and shows an appropriate message.
If PID number provided is not in the range(1 to 32768) it will return invalid argument error(EINVAL)
If the user input is negative the user will get an error message.
And if the input value by the user is greater than 100 the program will terminate.
Here i am restricting the user input at 100 as my system stops working if i try giving a value greater than 100, also there are many problems to show the clear working code, as it is not necessary that every time the system will take equal time for the same process to execute.I even noticed that there is a drastic difference between the execution time of the parent and child process.
Hence the execution of code is not very predictable and the process with syscall may sometimes take more time than the process without it.

Sample test case:
Enter any value between 0-100;