WRITE UP FOR QUES 1 - ASSIGNMENT 3

SUYASHI SINGHAL
2019478
CSE

Description of my code and implementation of my system call

Basic idea

- Basically in this assignment we have to add soft real time requirement to a process whose pid is given in the system call. We need to give x units of timeslice to each process that requires soft real time guarantees. We need to give higher priority to a processes softrealtime requirement compared to the vruntime that is usually considered.
- In order to implement this in our kernel we are adding a field rtnice(soft real time value) into the sched_entity struct. It has an initial value as 0.
- We are then using a system call of the same name - rtnice in order to update the rtnice value for a given process.
- We have also made certain changes in the cfs scheduler code to give more priority to a non zero soft real time value of a process.
- In the scheduler - more priority is assigned to a process with soft real time values than vruntime. Moreover, if more than one process has non zero rtnice value, then the one with lesser value of rtnice is given more priority.

1) ***Description of my system call - logical and implementation details***

- The name of my system call is “rtnice”. It takes two parameters as input. One is the processID (integer) for which we want to update the rtnice(soft real time value) and the other is the realtime (long) which is the rtnice value to be updated for the given process.
- In the system call we use the function find_get_pid() which takes the pid as parameter to get the struct pid. In the pid_task() function , I give the struct pid(obtained from find_get_pid()) and PIDTYPE_PID as the parameters. The function returns the task_struct structure which is then used to update the value of rtnice in struct sched_entity (which is one of the variables in task_struct).
- In case the find_get_pid() returns NULL, it means the process does not exist. In this case, I print a kernel alert along with the process id that the process does not exist in kernel log using printk(). I also print the error

message that “Process ID cannot be negative” on kernel log in case the user enters a negative pid. I then return the -ESRCH which means that the process does not exist. I print the errors on the console using strerror(errno) in the test.c file since the errno is set when the error is returned by the system call.

- In case the process is a valid one i.e find_get_pid() does not return null, I find the task_struct for that process and update the value of rtnice in sched_entity structure as realtime*50000000000.
- In case the rtnice value given by the user in the parameter of system call is negative, I do not update the value of rtnice. I print the error “Soft real time value cannot be negative” on the kernel log. I then return the error -EINVAL which stands for invalid arguments. It sets the errno and thus I can print the error on the console by strerror() function using the test.c file.
- In case the rtnice value was updated, I print the updated rtnice value on the kernel log and the corresponding pid of the process.

2) ***Changes done in the kernel code to write the system call and add the variable rtnice(real time guarantees) in task_struct for every process***

a) *In the directory include/linux/ I have edited the file sched.h and added the unsigned rtnice long variable as part of the sched_entity struct of the process using the line*
u64 rtnice;

b) *In the directory include/linux/ I have also edited the file core.c to add the soft real time requirement rtnice into the sched_entity struct of task_struct and initialize it’s value to 0 using the following line -*
p->se.rtnice =0;

c) ***In the directory kernel/sched/ I have edited the file fair.c which is the file for the CFS scheduler in order to incorporate the soft real time guarantees in the scheduling. I have made the following changes in the below mentioned functions -***

1) ***update_curr***
Change -

```
schedstat_add(cfs_rq->exec_clock, delta_exec);

if(curr->rtnice == 0)
{
    curr->vruntime += calc_delta_fair(delta_exec, curr);
}
else
{
    curr->rtnice = ((curr->rtnice <= delta_exec)? 0 : curr->rtnice - delta_exec);
    return;
}
update_min_vruntime(cfs_rq);
```

Logic - *If a process has a non zero value of rtnice, then we update the rtnice value. If the rtnice value is greater than delta_exec(time for which process executed) then the rtnice value is decremented by delta_exec. Otherwise it is set to 0.*
In case rtnice is 0 already, then scheduling occurs according to vruntime and hence vruntime value is updated. Only one of the rtnice or vruntime value is updated.

2) *entity_before -*
   Change -

```
static inline int entity_before(struct sched_entity *a,
                struct sched_entity *b)
{
    if(a->rtnice !=0 || b->rtnice !=0)
    {
        if(a->rtnice > 0 && b-> rtnice >0 )
            {
                if(a->rtnice > b->rtnice)
                    return 0;
                else if(a->rtnice < b->rtnice)
                    return 1;
            }
        else if(a->rtnice == 0 && b->rtnice >0 )
        {
            return 0;
        }
        else if(b->rtnice == 0 && a-> rtnice >0)
        {
            return 1;
        }
    }
    return (s64)(a->vruntime - b->vruntime) < 0;
}
```

Logic *- In this function we are giving more priority to a process which has a smaller rtnice value(non zero) as compared to rtnice value of the other process (given both have non zero rtnice values). If one of them them has rtnice value as 0 then the other gets more priority as it has a nonzero soft real time requirement.*

Changes for adding system call -

a) I have made a directory modifyCFS in the linux directory which stores my system call and makefile. I added two files into the directory. The first is my system call i.e rtnice.c and the other is the Makefile. The makefile contains a single line i.e -
obj-y := rtnice.o

***The directory also contains the actual system call c file i.e rtnice.c**

b) In the Makefile inside the linux directory I have made a change in one of the lines as follows -
core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ modifyCFS/

*I added the **modifyCFS/** into the above line so that the files in the directory modifyCFS can also be compiled during compilation.*

c) *In the directory arch/x86/syscalls/ I have edited the file syscalls_64.tbl and added the following line at the end -*
440 common modifyCFS sys_rtnice

Here,
440 is the system call number
common is the type of system call
modifyCFS stores the system call and corresponding Makefile
sys_rtnice is the name of the system call.

d) *In the directory include/linux/ I have edited the file syscalls.h and added the following line at the end -*
asmlinkage long sys_rtnice (pid_t processID, long realtime);

e) *Finally after making all the changes I compiled the kernel, installed it and rebooted the virtual box to add my system call.*
I used the following commands -
sudo make localmodconfig
sudo make
sudo make modules_install install
sudo make install
shutdown -r now

Testing the system call

Description of my test.c

I have added two test functions in test.c The user can choose on of them by entering either 1 or 2

1) ***Test function 1***

   Description of code

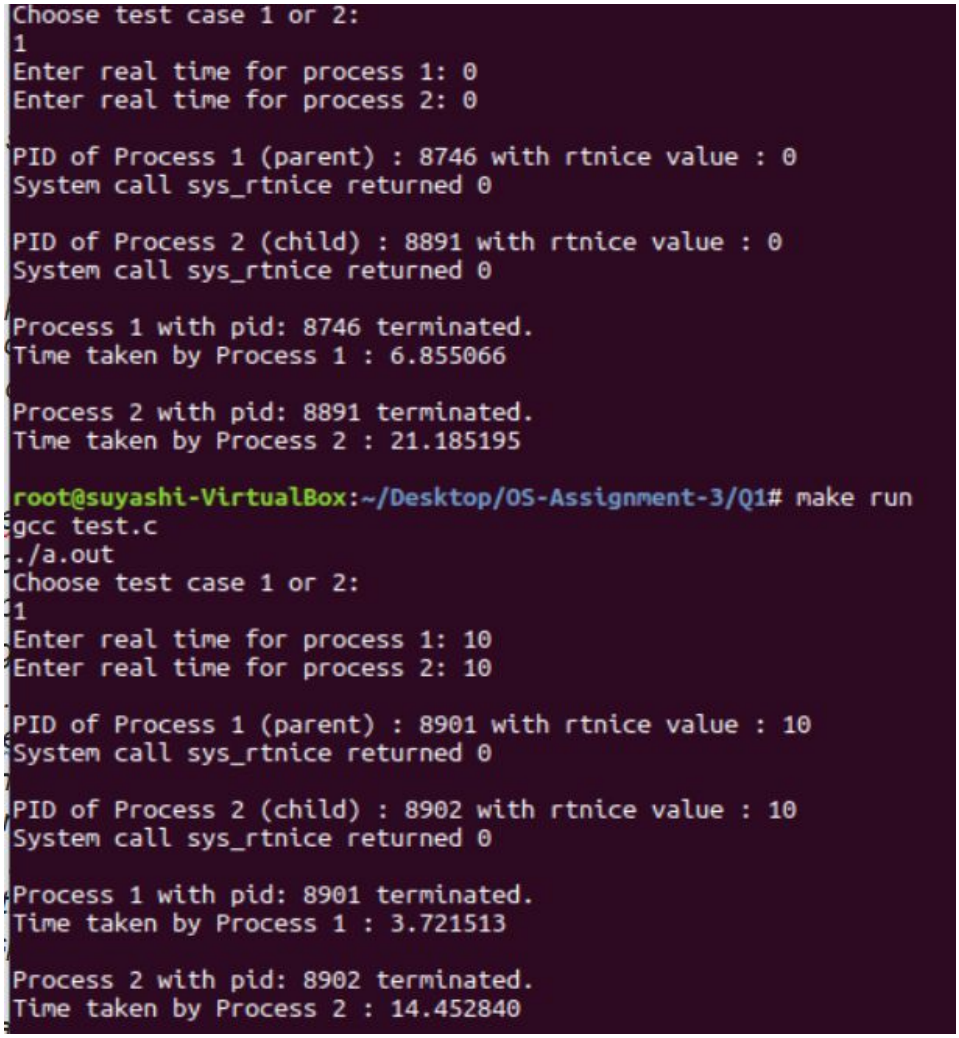
   - *The function first prompts the user to enter two values of rtnice for process 1 and process 2 respectively.*
   - *Process 1 is parent and process 2 is child process.*
   - *If the values of rtnice are negative then the syscall returns -1 and print "invalid arguments error".*
   - *We first fork a child process . In the parent process(process 1), we print the pid of the parent and it's rtnice value entered by the user. We then call the system call rtnice which takes two arguments. First argument is the pid of the process 1 that we get by using getpid()and the second argument is soft_rtnice1 entered by user. We then print the value returned by the system call. In case it returns -1, we print the corresponding error using strerror() function.*
   - *Similarly we print the pid and rtnice value of the process 2 (child) and call the system call for the child within the parent itself. First argument is the pid of the process 2 that we get by using pid variable and the second argument is soft_rtnice2 entered by user. We then print the value returned by the system call. In case it returns -1, we print the corresponding error using strerror() function.*
   - *After that within the parent we execute a large function (large_factorial) measure its time using clock_gettime() function. When the large function completes execution, we print the pid of the parent and the time taken by it. Moreover the parent waits for the child to exit.*
   - *Within the child process we once again execute a large function which takes around 4 times more time as compared to that in parent (Since we have entered a larger value of the parameter) Again using clock_grttime() function we measure the time of execution and then print the pid of the child and the time taken by it.*

Input the user should give

- *The system call takes rtnice value and pid as user inputs.*
- *In case of pid, if pid of a process that does not exist is entered, then the system call returns -1 and prints the error “No such process” using strerror(). Moreover if a negative pid is entered by the user, then an error is displayed on the kernel log.*
- *The user should give a non zero rtnice long as input.*
- *In case the user enters a string input (i.e non integral input) for rtnice value , then the error has been handled and we display “I/O error” (errno = 5) .*
- *Also, negative value of rtnice is not allowed to be entered. If the user enters a negative value of rtnice for either process 1 or process 2, then the corresponding system call returns -1 and displays the error for EINVAL - “Invalid arguments” using strerror().*
- *Hence the user has to enter an integer pid and long rtnice. The rest is handled by the system calls which returns 0 if the values entered are valid. Else, it returns -1 and sets the errno for the corresponding error which we then display on the terminal.*

Expected output

a) ***Comparing processes with and without soft real time requirements*** *- In the given screenshot output , we can see that when the child and parent are given equal non zero rtnice value each (soft_rtnice1=10 and soft_rtnice =10), the time of execution for each process is less than in the case when both parent and child have 0 as their rtnice value. This means that the time taken to execute the processes is more without the soft real time requirements as compared to when processes are given soft real time requirements.*



b) ***When process 1 is given more priority*** *- In the given output , we have given more priority to the process 1 (parent) as compared to process 2 (child) since rtnice value for the process 1 (soft_rtnice1 =1)is lesser than that in process 2(soft_rtnice2 = 1000). Hence the parent completes it's execution first as compared to the child. This will also happen when both are given 0 rtnice values as execution time of process 1 is less than that of process 2 in general.*

```
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-3/Q1# make run
gcc test.c
./a.out
Choose test case 1 or 2:
1
Enter real time for process 1: 1
Enter real time for process 2: 1000

PID of Process 1 (parent) : 8913 with rtnice value : 1
System call sys_rtnice returned 0

PID of Process 2 (child) : 8915 with rtnice value : 1000
System call sys_rtnice returned 0

Process 1 with pid: 8913 terminated.
Time taken by Process 1 : 3.610048

Process 2 with pid: 8915 terminated.
Time taken by Process 2 : 14.225846
```

c) ***When process 2 is given more priority*** *- In the given output , we have given more priority to the Process 2 (child) as compared to Process 1 (parent) since rtnice value for the Process 1 is more than that in Process 1. Hence the child completes it's execution first.*
Even tough the time of execution of child is 4 time than that in parent (as a function that takes 4 times more time is executed) , the child completes it's execution first, since it has more priority.

```
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-3/Q1# make run
gcc test.c
./a.out
Choose test case 1 or 2:
1
Enter real time for process 1: 1000
Enter real time for process 2: 1

PID of Process 1 (parent) : 8922 with rtnice value : 1000
System call sys_rtnice returned 0

PID of Process 2 (child) : 8923 with rtnice value : 1
System call sys_rtnice returned 0

Process 2 with pid: 8923 terminated.
Time taken by Process 2 : 14.108151

Process 1 with pid: 8922 terminated.
Time taken by Process 1 : 3.591754
```

Error handling

a) ***When rtnice value is entered as a string instead of long*** *- In the given output , I have entered a string value in the rtnice variable instead of long.*

Hence an error is displayed on the terminal " I/O error" using strerror() function. “EIO” is the error macro for the same(errno = 5).

```
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-3/Q1# make run
gcc test.c
./a.out
Choose test case 1 or 2:
1
Enter real time for process 1: qwer
Error: Input/output error
```

```
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-3/Q1# make run
gcc test.c
./a.out
Choose test case 1 or 2:
1
Enter real time for process 1: 10
Enter real time for process 2: erjer
Error: Input/output error
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-3/Q1#
```

```
if(scanf("%ld", &soft_rtnice1) != 1)
    {
        printf("Error: %s\n", strerror(EIO));  // EIO (errno 5)
        return 0;
    }
printf("Enter real time for process 2: ");
if(scanf("%ld", &soft_rtnice2) != 1)
    {
        printf("Error: %s\n", strerror(EIO));  // EIO (errno 5)
        return 0;
    }
```

b) *When a negative rtnice value is given by the user -*

In the given output, we have entered a negative value for the rtnice variable of the process 2. Hence when the system call executes for process 2, it returns -1 and prints the error on the terminal "Invalid arguments". This is because in case of negative rtnice values, the system call returns -EINVAL which sets the errno and hence we print the error using strerror() function.

```
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-3/Q1# make run
gcc test.c
./a.out
Choose test case 1 or 2:
1
Enter real time for process 1: 10
Enter real time for process 2: -100

PID of Process 1 (parent) : 8938 with rtnice value : 10
System call sys_rtnice returned 0

PID of Process 2 (child) : 8939 with rtnice value : -100
System call sys_rtnice returned -1

Error : Invalid argument
```

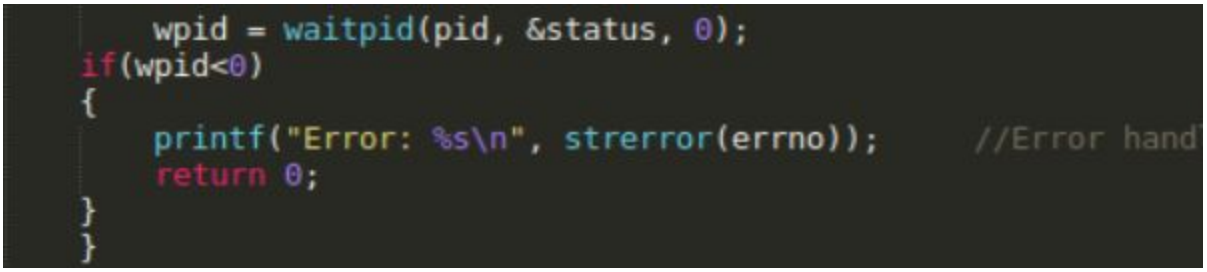
c) ***When the pid entered in the input of system call does not exist*** - *In this case I have altered the test.c file to show that the error for negative pid has been handled in system call.*

```
Enter real time for process 1: 4
Enter real time for process 2: 4

PID of Process 1 (parent) : 9011 with rtnice value : 4
System call sys_rtnice returned 0

PID of Process 2 (child) : 113162 with rtnice value : 4
System call sys_rtnice returned -1

Error : No such process
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-3/Q1# PID of Process 1 (parent) : 9012 with rtnice val
ue : 4
System call sys_rtnice returned 0

PID of Process 2 (child) : 113162 with rtnice value : 4
System call sys_rtnice returned -1

Error : No such process
```

d) ***Error handling for fork() in first test function***

```c
pid = fork();
if(pid<0)                    // Error handling for negative pid
{
    printf("Error: %s\n", strerror(errno));
    return 0;
}
else
{
```

e) ***Error handling for waitpid() function in test function 1 -***



```
    wpid = waitpid(pid, &status, 0);
if(wpid<0)
{
    printf("Error: %s\n", strerror(errno));      //Error hand
    return 0;
}
}
```

2) ***Test function 2***

Description of code

- *In the second test function I have first printed the execution time for a given large function for 5 processes* ***without using soft real time requirements****. I have used a while loop that forks 5 child processes and prints the time of execution for each process. The time of execution is calculated using the clock_gettime() function and displayed on the terminal.*
- *After this, we print the execution time for the same large function for 5 processes that have been* ***given different non zero soft real time requirements****. I have used a while loop that forks 5 child processes and prints the time of execution for each process. Within the while loop the system call rtnice is called for each process to set the soft real time requirements. The time of execution is calculated using the clock_gettime() function and displayed on the terminal for each process.*
- *We have handled the error for fork() in test.c and rest of the errors have been handled in the system call as displayed above.*

Input the user should give

- *The system call takes rtnice value and pid as user inputs.*
- *In case of pid, if pid of a process that does not exist is entered, then the system call returns -1 and prints the error "No such process" using strerror(). Moreover if a negative pid is entered by the user, then an error is displayed on the kernel log.*
- *The user should give a non zero rtnice long as input.*
- *In case the user enters a string input (i.e non integral input) for rtnice value , then the error has been handled and we display "I/O error" (errno = 5) .*
- *Also, negative value of rtnice is not allowed to be entered. If the user enters a negative value of rtnice for either process 1 or process 2, then the corresponding system call returns -1 and displays the error for EINVAL - "Invalid arguments" using strerror().*

- *Hence the user has to enter an integer pid and long rtnice. The rest is handled by the system calls which returns 0 if the values entered are valid. Else, it returns -1 and sets the errno for the corresponding error which we then display on the terminal.*
- *In this test case, we have hardcoded the test case. Hence the user does not need to enter anything into the terminal.*

Expected Output

In the output screenshot given below, we can see that the execution time when soft real time requirements are set is much smaller as compared to when soft real time requirements aren't set.

```
root@suyashi-VirtualBox:~/Desktop/OS-Assignment-3/Q1# make run
gcc test.c
./a.out
Choose test case 1 or 2:
2
Time taken to run 5 processes without soft real time requirements:
Process 3 took : 3.126925 seconds.
Process 2 took : 3.149440 seconds.
Process 1 took : 3.153611 seconds.
Process 5 took : 3.185789 seconds.
Process 4 took : 3.196017 seconds.
Time taken to run 5 processes with soft real time requirements:
Process 5 took : 0.783450 seconds.
Process 4 took : 0.716257 seconds.
Process 3 took : 0.721019 seconds.
Process 2 took : 0.746480 seconds.
Process 1 took : 0.742030 seconds.
```

Error Handling

a) *We have shown all the error handling done in test case 1*
b) *Error handling for fork()*

```
while(count<5)
{
    pid = fork();
    if(pid<0)
    {
        printf("Error: %s\n", strerror(errno));
    }
```

Generation of diff

- *I have generated the diff of my linux directory with the original linux directory.*
- *I have two directories in /usr/src -* ***my_Scheduler*** *which stores the modified code of linux-5.9.1 in which I have added my system call rtnice and* ***original*** *which stores the original code of linux-5.9.1*
- *Before generating diff I used the command* ***sudo make distclean*** *in my_Scheduler/linux-5.9.1/ directory.*
- *After that I generated the diff using the command -* ***diff -Naur original/linux-5.9.1/ my_Scheduler/linux-5.9.1/ > diff.txt***