CSE 231: OS Assignment 3

Modifying CFS Scheduler

1 Description and implementation of the code

1.1 System Call

- SYSCALL_DEFINE2 was used as 2 input parameters were passed
- The two parameters were the pid of the required function and the soft real time requirement value to be used
- It was checked whether the input values of pid and the soft real time requirement values were valid or not
- The value of pid was valid only when there is an existing process with the given pid
- The values of the soft real time values are valid only when they are comparable with the already existing vruntime and delta_exec values of the process and at the same time are in the range of u64
- To ensure the soft real time requirement values are well within range, the user can enter a value between 0 and 30000000000.
- The value entered will be multiplied by a fixed value in the code so that the value is acceptable for the code.
- If 0 is entered as the value, the program will run as if there is no restriction on selection of the processes.
- Then all the processes are compared and the required process is assigned the required soft run time requirement value.

```c
SYSCALL_DEFINE2(rt_nice, long, pid, long, realtimeRequirement)
{
    bool isPresent = false;
    struct task_struct *task;
    if(realtimeRequirement < 0 || realtimeRequirement > 30000000000)
    {
        printk("Invalid soft time value");
        return 1;
    }
    realtimeRequirement *= 2500000001;
    for_each_process(task)
    {
        if(pid == (long)task->pid)
        {
            isPresent = true;
            printk("Changed rt_nice value of: %s", task->comm);
            task->se.rt_nice = realtimeRequirement;
        }
    }
    if(isPresent == false)
    {
        printk("Invalid PID!");
        return 1;
```

1.2 fair.c

- Two functions have been modified in fair.c so that the required changes can be made.
- The functions are:
  - static void update_curr(struct cfs_rq *cfs_rq)
  - static inline int entity_before(struct sched_entity *a, struct sched_entity *b)

1.3 test.c

- The program after taking the input value of soft real time requirement values, forks the process.
- In the parent process a loop is executed 10000000 times.
- Time values just before the start of execution and just after the execution are noted and the time difference is displayed.
- In the child process a loop is executed 100000000 times.
- Time values just before the start of execution and just after the execution are noted and the time difference is displayed.
- The single difference is just the system call is called before the start of loop, hence giving the current process more priority.

```c
printf("Enter the requires realtime value(between 0 and 30000000000): ");
scanf("%lld", &realtimeValue);
if((pid = fork()) == 0)
{
    long ret = syscall(336, getpid(), realtimeValue);
    if(ret != 0)
    {
        printf("%s\n", strerror(ret));
        exit(ret);
    }
    gettimeofday(&t1, NULL);
    for(ctr = 0; ctr < 100000000; ctr++);
    gettimeofday(&t2, NULL);
    printf("Execution time of child = %lf milliseconds for ctr = %lld\n", (doub
}
else if(pid > 0)
{
    gettimeofday(&t1, NULL);
    for(ctr = 0; ctr < 10000000; ctr++);
    gettimeofday(&t2, NULL);
    printf("Execution time of parent = %lf milliseconds for ctr = %lld\n", (dou
    wait(NULL);
}
```

2 Inputs from the user

- The user is required to just give a single input of the soft real time requirement values.

```
pw@ubuntu:~/Desktop/Sem3/OS/Assignment3$ make run
gcc test.c
./a.out
Enter the requires realtime value(between 0 and 30000000000): 1931
Execution time of child = 269.028000 milliseconds for ctr = 100000000
Execution time of parent = 298.871000 milliseconds for ctr = 10000000
```

3 Outputs

- In the above pic, the outputs of the program can also be seen.
- Along with this, the statements can also be seen on kernel using dmesg | tail command

```
pw@ubuntu:~/Desktop/Sem3/OS/Assignment3$ dmesg |tail
[ 2379.202678] Changed rt_nice value of: a.out
[ 2437.203912] Changed rt_nice value of: a.out
[ 2441.133755] Changed rt_nice value of: a.out
```

4 Error Handling

- If everything runs fine, then 0 is returned.
- If there is an error, 1 is returned and that process is terminated.
- The error can be mainly due to the below reasons:
  - There is some issue while forking the processes
  - The pid doesn’t correspond to any currently available process
  - The value of soft real time requirement value inputted is out of the previously mentioned range.
- The errors corresponding to the above points can also be viewed in the kernel using dmesg.

5 Diff File

- The diff file was created after running the command make clean
- To create the diff file, “-r”, “-u”, and “-N” flags were used.

6 Sample test cases

- The program can be run by taking any value between 0 and 30000000000