Operating Systems

Assignment-3

Rohan Jain
2019095

Question 1

Description

The system call `rtnice` sets the value of `soft_realtime_nice` of the `task_struct` corresponding to the given PID. The CFS scheduler now schedules tasks based on their `soft_realtime_nice` values and then on the basis of their `vruntime`.

Function definition

```
long sys_rtnice(pid_t pid, long long srt_val);
```

Returns 0 on success and negative number on error. It sets `errno` on return.

Explanation of Code:

Explanation of System Call rtnice:

```
SYSCALL_DEFINE2(sh_task_info, pid_t, pid, long long, srt_val)
```

This is the main system call function that is defined using the `SYSCALL_DEFINEn` macro. If the arguments `pid` or `srt_val` have negative values the system call returns `-EINVAL` and `errno` is set to `EINVAL` = invalid arguments. `srt_val` expects the soft real time requirements to be in nanoseconds.
To find the `task_struct` corresponding to the `pid` passed in parameter I have used functions `find_vpid()` and `pid_task()`.

```
task = pid_task(find_vpid(pid), PIDTYPE_PID);
```

If the `task_struct` corresponding to the `pid` is not found, the `errno` is set to `ESRCH` = No such process and the returned value is `-ESRCH`.

The `soft_realtime_nice` value of the process is changed by the command:

```
task->se.soft_realtime_nice = (u64)(srt_val);
```

The function returns 0 if everything works fine.

Explanation of changes for CFS scheduler:

The first change is in the `struct sched_entity` in `include/linux/sched.h`. I add a variable `u64 soft_realtime_nice` that corresponds to the soft real time requirements of the process.

The second change is in `kernel/sched/core.c`, The initial value of `soft_realtime_nice` is set in `__sched_fork()` function and is set to `0`.

Third changed file is `kernel/sched/fair.c`, the functions I changed were `entity_before()` which is a comparator for the rbtree and the function `update_curr()` which updates the value of the `soft_realtime_nice` of that process. If the functions fail to compare between the `soft_realtime_nice` values of the processes then the comparison is done on the basis of their `vruntime`.

Note: If a process has realtime requirements of 5 sec and the other process has a real time requirement of 10 sec, then the 5 sec one is going to get the priority because that task is more time sensitive.

Expected Input Output

The user has to call the system call and give 2 inputs `pid_t pid` and `long long srt_val`. `pid` refers to the pid of the process whose details have to be printed and `srt_val` refers to the soft real time requirements of the process in nanoseconds.

All errors returned by code:

`ESRCH`
`EINVAL`

The conditions for their return are explained above.

Test Cases

Sample Output:

```
rohanj-02@ubuntu:~$ ./test3
Enter soft realtime requirements: 120
Time taken with soft realtime requirements of 120000000000: 21.022342
Time taken without soft realtime requirements: 40.656639
rohanj-02@ubuntu:~$ ./test3
Enter soft realtime requirements: 0
Time taken with soft realtime requirements of 0: 44.746420
Time taken without soft realtime requirements: 44.813979
rohanj-02@ubuntu:~$ ./test3
Enter soft realtime requirements: -10
rtnice: error: Invalid argument
Time taken without soft realtime requirements: 20.496647
rohanj-02@ubuntu:~$ ./test3
Enter soft realtime requirements: 20
Time taken with soft realtime requirements of 20000000000: 20.410283
Time taken without soft realtime requirements: 39.851356
rohanj-02@ubuntu:~$
```

In case of 120 soft real time requirements, the process runs first and the other process waits for its turn. When the requirements are 0, both processes take the same amount of time to run. To show error handling, I sent a requirement of -10. To show `ESRCH`, one can change the code and change the `pid` variable to some invalid pid. In the last case also, just like the first case the priority is visible as the first process gets completed in 20s and the second process takes 39 sec to complete.

To test `ESRCH`, send an invalid pid like a negative number or a really large number.
To test `EINVAL`, send the soft real time requirements as negative or a negative pid.

Running test.c

In the testfile, the process is forked after the input has been taken and the real time requirements of the process is set using the syscall `rtnice`. After that both run a loop of 10^{10} and display the time taken to run that loop.