CSE231 - Operating Systems

Assignment-3

Name: Ananya Lohani
Roll Number: 2019018
Branch: CSE

Description:

In this assignment, I have modified the **sched_entity** structure to contain a new field **rt_nice** which is the soft real-time requirement of a process(a value of x as **rt_nice** means that the process must receive at least x units of time-slice). To give priority to processes with soft real-time requirements I have modified the CFS scheduler to schedule tasks on the basis of **rt_nice** first, and **vruntime** second.

rt_nice System Call implementation:

I added a new field **rt_nice** to the **sched_entity** struct in **/kernel/sched/sched.h** and initialised it to 0 in the **__sched_fork()** function in **/kernel/sched/core.c**. I have added a system call **rt_nice** which takes two arguments: **PID of the process** and the **soft real-time requirement**. The system call gets the task corresponding to the given PID and sets the **rt_nice** field of its **sched_entity** to the given value. The system call number of sys_rt_nice is 441.

Modified CFS implementation:

The CFS scheduler of Linux has been modified to give priority to the **rt_nice** values before giving priority to the **vruntime** of processes in the run-queue.

Functions modified:-

- **entity_before():** This function acts as a comparator between the vruntime of 2 **sched_entity** struct pointers. I added the code that compares the **rt_nice** values of the pointers first, and if both pointers' **rt_nice** is equal to 0, it compares their **vruntime**.
- **__pick_next_entity():** This function picks the next process to be executed. I modified it to check if any process in the runqueue has a non-zero rt_nice value and selected the process with the minimum

rt_nice to be executed. If no process has **rt_nice** greater than zero, it will execute the next process according to **vruntime**.

- **update_curr():** If the **rt_nice** value of a **sched_entity** is greater than 0, it updates **rt_nice** by subtracting from it the amount of time that the process ran.

Errors handled:-

Errors returned by the system call are handled by **perror()** in **test files**.

- **EINVAL**: If an invalid pid was given(pid not between 1 and 2147483647) or if invalid value for **rt_nice** was given(<0).
- **ESRCH**: If there's no process corresponding to the given pid.

Testing the Scheduler:

The working of the scheduler can be tested by running **test-cases/test.c**. The program forks 5 processes, executes a loop of 2000000000 and displays the time taken in two cases: when there are no soft real-time requirements(**rt_nice = 0**) and when there are non-zero soft real-time requirements(**rt_nice > 0**). The file **test-cases/rt_nice.c** included in **test.c** calls the system call **rt_nice** and supplies the pid and soft real-time requirements as the argument.

Expected output:

The processes with rt_nice > 0 take **less time** than processes with rt_nice = 0. The expected output is as follows:

```
./test
Time taken with rt_nice = 0:
Process 1, PID: 1318, Time: 6.770956s
Process 2, PID: 1319, Time: 6.795187s
Process 3, PID: 1320, Time: 6.850028s
Process 4, PID: 1321, Time: 7.409110s
Process 5, PID: 1322, Time: 7.560659s
Time taken with rt_nice > 0:
Process 2, PID: 1332, rt_nice: 20, Time: 5.662679s
Process 3, PID: 1333, rt_nice: 30, Time: 5.775768s
Process 1, PID: 1331, rt_nice: 10, Time: 5.788870s
Process 5, PID: 1335, rt_nice: 50, Time: 5.831745s
Process 4, PID: 1334, rt_nice: 40, Time: 5.625402s
```

Note: This modified CFS would only work 80-90% of the time because there may be page faults and cache misses that cannot be controlled by the scheduler.