Writeup for Q1

Steps followed for compiling the kernel and adding system call:

1. Download kernel source code using wget command

wget

https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.17.4.tar.xz

2. Extract the tar.gz file using

sudo tar -xvf linux-4.17.4.tar.xz -C/usr/src/

3. Goto to directory kernel/ in the extracted folder

cd /usr/src/linux-4.17.4/

4. Edit file sys.c, ex : sudo nano sys.c

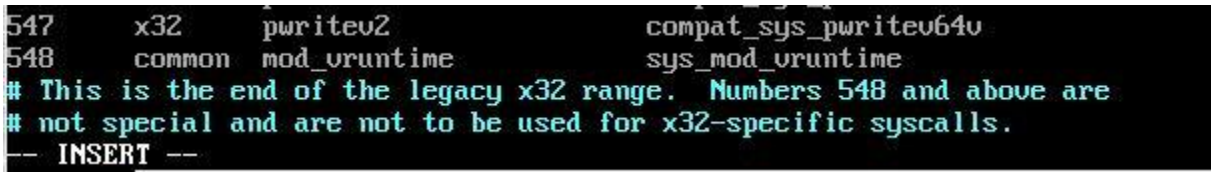
```
 */
SYSCALL_DEFINE2(mod_vruntime, int, p_pid, long, time )
{
        pid_t pid = p_pid;
        struct pid *pid_struct;
        struct task_struct *task;

        pid_struct = find_get_pid(pid);
        task = pid_task(pid_struct, PIDTYPE_PID);

        if(task == NULL){
                printk("Process not found\n");
                return -ESRCH;
        }

        task->se.vruntime += time;
        printk("Process %d's vruntime changed successfully ",p_pid);

        return 0;

}
```

5. Add your syscall code in the file sys.c

6. Go to directory arch/x86/entry/syscalls/ in the extracted folder

7. Add your system call to file syscall_64.tbl

Ex: 440 common sample sys_sample



8. Edit config file so that only required things are compiled

sudo make menuconfig

9. Compile the kernel with sudo make -j($number of processors to use)

sudo make -jn

10 .Install modules using : sudo make modules_install install

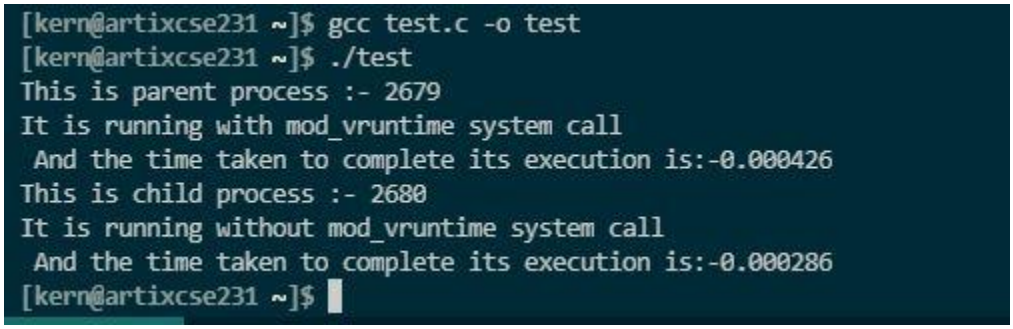
sudo make modules_install install

11. Reboot system

sudo reboot

Name of my syscall :- mod_vruntime();
short for modify runtime

The output of the test.c file



Explanation: -

I have created two processes using fork() i.e child and parent. The child process is executing without mod_vruntime syscall and the parent process is running with mod_vruntime.

We can clearly see that child process took:- 0.000286 sec
While parent process took:- 0.000426 sec

As we know that process having min vruntime are selected by the schedule first in the CFS algorithm.

If we increase the vruntime of the process using sched_entity structure then we can delay its selection because when we increase the vruntime then there will be self-balancing in the RB-tree and its execution time will be delayed.

Data structures used are :- struct in c

To achieve this:-

The definition of struct sched_entity is in file /include/linux/sched.h.

Now we change the actual functionality of the CFS scheduler. The CFS scheduler used a red-black tree to store and retrieve information about all the processes. By default the red-black tree picks the process with minimum vruntime.

This is achieved by changing the implementation of the function update_curr() in file kernel/sched/fair.c.

Now the next step is defining the System Call called rt_nice to change the value of vruntime of a process. In the system call all the information about the process is obtained using its pid in a task_struct using pid_task() and find_get_pid().

Error Handling Done in System Call :

2. Handling the case when the process is not found Code :

```
if (task == NULL) {
        printk("Process not found\n");
         return -ESRCH;
 }
```

Relevant structures used:-

```
struct sched_entity {
    struct load_weight  load;       /* for load-balancing */
    struct rb_node      run_node;
    struct list_head    group_node;
    unsigned int        on_rq;

    u64         exec_start;
    u64         sum_exec_runtime;
    u64         vruntime;
    u64         prev_sum_exec_runtime;

    u64         nr_migrations;

#ifdef CONFIG_SCHEDSTATS
    struct sched_statistics statistics;
#endif

#ifdef CONFIG_FAIR_GROUP_SCHED
    struct sched_entity *parent;
    /* rq on which this entity is (to be) queued: */
    struct cfs_rq       *cfs_rq;
    /* rq "owned" by this entity/group: */
    struct cfs_rq       *my_q;
#endif
};
```

Sched_entity defination