CSE 231: Operating Systems
Assignment 3
Modifying CFS Scheduler

Aditi Sejal
2019228

Description of Code

I have introduced a parameter called sruntime in the sched_entity struct present in the task_struct struct in the linux kernel file sched.h. This sruntime refers to the soft runtime and is the soft real time guarantee to be provided to the process. It is zero by default and can take only positive values. General range of values is 10^8 - 10^9 since the time is taken in nanoseconds. An example of valid value for this could be the integer 1,00,00,00,000. For a process with sruntime equal to this, it will have higher priority over a process with 0 sruntime value and will thus finish quicker without having to wait for processes with lower priority. Consider the case of two processes, processA with sruntime = 10,00,00,000 and another processB with sruntime = 9,00,00,00,000. Here process B will have higher priority and thus will be executed before process A which has lower priority considering their sruntime values.

In the fair.c file, which has the implementation of the CFS scheduler, I have added a few lines to make sure the sruntime of the current process is also updated along with the other fields in the update_curr function. The update_curr function is responsible for updating the current task's runtime statistics, updates the vruntime and now also sruntime, then calls account_cfs_rq_runtime which in turn eventually calls resched_curr which will fix the red black tree. In case the value of sruntime for current process is positive, we decrement it by delta_exec or equate it to zero (whichever gives maximum non-negative value) which is the time for which current process has run after the last update of its parameters.

I have also made changes in the entity_before function, which is responsible for returning values 0 or 1 according to if sched_entity 'a' and sched_entity 'b', its two parameters, 'a' should be scheduled before 'b'. To make sure, sched_entity along with the already present check on vruntime we also look at the sruntime values for them, I have added a few if-else statements which return 1 if a has a higher value of sruntime than b and 0 otherwise. This is so because higher value of sruntime means higher priority and that this process must be scheduled first and thus move towards the left of the red black tree to be picked first. Thus the pick_next_entity function which calls entity_before will now correctly choose the process with higher soft real time requirements.

I have also made a system call called rtnice which takes two parameters, integer type pid and integer type rtg (real time guarantee) and sets the sruntime value of the process with given pid

to rtg. The implementation for this starts with a PID struct corresponding to the integer PID passed as parameter which is obtained using the find_get_pid function which takes the PID (integer or pid_t type) as input. This PID struct is used to get the corresponding task_struct which is a structure, an interface to the scheduler, used to store details of processes in the linux kernel. This is done using the pid_task function which takes the pointer to the PID structure and returns the corresponding task_struct for the process with the given pid. In case no process with
given pid exists, an error number is returned accordingly.The task_struct has fields which describe the process one of which is 'se' of struct sched_entity type which is used to access sruntime and change it from default value of zero to that as of rtg (second parameter of syscall). The system call returns 0 on successful execution while it returns -1 on unsuccessful termination and sets global variable errno to the error number specified.

Expected Inputs

The user should give two parameters to the system call, the process PID and the real time guarantee - both of integer type. The PID of the process should be valid for successful execution of system call. The real time guarantee should be as defined in the first paragraph of the description of the code section.

Expected Outputs

The system call returns 0 on successful execution and otherwise returns the error number for the error and to which the global variable errno is also set. The interpretations of the error number as described below. The error number and corresponding error messages are printed using the perror in errno.h header file.

For comparing the real time guarantees for two processes, and proper working of the modified scheduler, a test program test.c has been written which prints the time it took for a computation (a large empty for loop in my case). This time has been fetched using the gettimeofday() function which gives the exact time for the moment and it is stored in the struct timeval. An empty for loop is run for some time after which the time is recorded again. The difference between the start and end times tells the time the process took to complete the execution which ideally should and mostly is lesser for the process with the soft real time guarantees. This can be observed around 85 per cent of the time while in the rest of the cases, we can safely assume that things like cache miss and page faults cause this. The CFS scheduler thus, works properly if the time taken for the process with higher soft real time guarantee is lesser as compared to that with lower soft real time guarantee.

Error Handling

The following errors are handled by the system call rtnice-

1. Error 3

ESRCH 3 /* No such process */
This means that the process with given pid does not exist and thus its corresponding task_struct can not be found by the kernel. Give PID of an existing process to resolve the error.

2. Error 22

EINVAL 22 /* Invalid argument */
All PIDs for processes are greater than 0 and have upper limit as pid_max value which is used as a safe bound in the syscall implementation. If this error occurs provide a value between o and 131072 to resolve it. This error can also occur if negative value for soft real time requirements is given to a process. To rectify this, check that a valid value (non-negative integer) has been given to the soft real time guarantee, the second parameter of syscall.

Testing the System

Three test files have been provided to check all cases for this modified scheduler.

1. test.c
   This checks the most basic functionality of the system, to check if a process with real time requirements is given higher priority and executed before one with no such requirements. The time taken for the latter which is the normal process should be less than that for the former. Can be run using the 'make run' command on cmd in the correct directory.
2. testrt.c
   This checks whether the process with lower real time guarantees is executed after one with higher real time guarantees. Both processes have positive values of sruntime but the one with greater value of sruntime shows lesser time taken.
3. testerr.c
   This does error handling of three types - invalid PID value, invalid soft real time guarantee value and incorrect PID value (process does not exist). The error number and corresponding error messages are printed using the perror in errno.h header file.