CS 423 MP2 Report

Group 23

Group Members: Xingbang Liu, Dajun Xu,, Ruqi Zhong
netID: xl14, dajunxu2, rzhong5

Architecture:

In this MP, we implemented a real-time scheduler on the basis of Rate-monotonic Scheduling(RMS). That is, the smaller the rate is, the higher the priority is of the task. We use a proc filesystem to read the status information of applications from user space to kernel space. We user the admission control to decide if an application can register in the proc file system. In our user application, every time we finish a job, which is a factorial operation, we send a yield signal to make the task sleep for a period before its state becomes ready for running. We reset the timer to control the sleep time of the task. The timer handler wakes up the task after the sleep time is out and wake up the dispatch function to schedule a new task to run. When an application registers, it passes its registration paramters such as PID, period and processing time to the kernel module. When the real-time while-loop has finished, the application will pass a de-registration message to the kernel to let the RMS scheuler know that it has finished.

For the timer handler function, we use the spin lock for that the execution of the timer handler can be regarded as the top half interrupt, which can not go to sleep. We use mutex for other locks of resources in the borrom half, which will make the thread go to sleep if the resource locked by mutex is not successfully called that thread.

Implementation:

1. highestPrior( ):

This function iterates through the proc list to find the task struct that has the shortest period and a READY state.

2. dispatch_thread( ):

This function takes the output from the highestPrior( ) function as input. If the input is NULL, then we just preempt the currently running task. Else we preempt the currently running task with the task with a higher priority.

3. Timer_handler(unsigned int task_pid):

This function iterates though the proc list to find the task with pid = task_pid. This function is called every period of each task registered to wake up the task. Then it wakes up the dispatch_thread( ) function to schedule a new task for running.

4. admission_control( ):

This function makes sure the CPU utilization is below a pre-set threshold. It will prevent new process from registering into the system if the CPU utilization is high.

5. registration( ):

This function initialize a task's pid, period, cpu_time, next_period, task_struct , state information. It also initialize the task's list_head and timer structs. And will add the new taks into the proc list.

6. yielding( ):

This function is called every time a task finishes its computation. It sets the task state from RUNNING to SLEEPING and sets the task's timer next wake-up time. It also wakes up dispatcher to schedule a new task for running.

7. de_registration( ):

This function is called at the kernel exit module to free space of every task in the proc list including the timer and the corresponding cache object.

8. mp2_write( ):

This function is called whenever there is a newly-created task. The task will register itself by writing its pid, cpu_time, period information into the proc file system entry.

Details of how to run the program:

$ make

$ sudo insmod mp2.ko

$ ./userapp1 period computation_time iteration_number
& ./userapp2 period computation_time iteration_number

Screenshot:

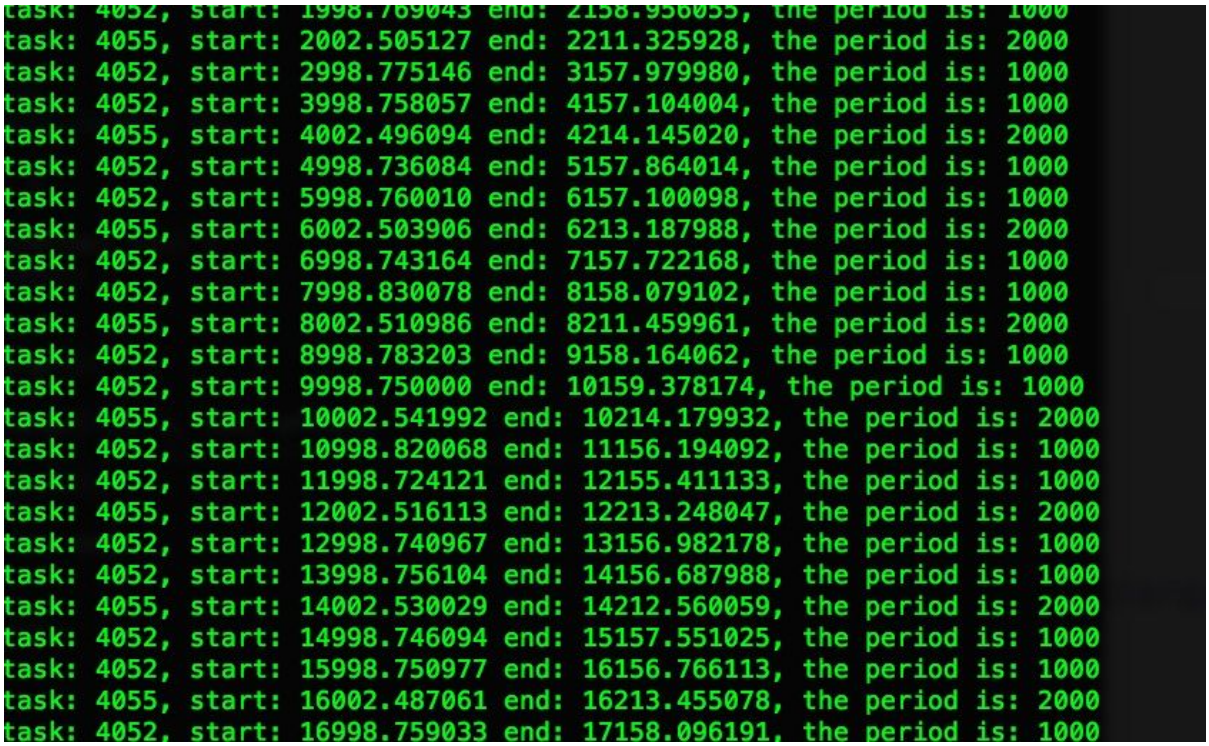

Figure 1 iteration 30000, cpu_time 159ms