Hao Zheng hzheng19
Zhili Feng zfeng6
Joo Won Lee jlee164

First, we got the basic skeleton from MP1.

We did some modification on the write function, so that it can parse register, yield and unregister message accordingly. We classified input by checking the first character of each string: R as register, Y as yield and D as unregister.

For “mp2.c”, we had a struct called ‘mp2_task_struct’. This struct contains a task_struct pointer, a wakeup timer, the process id, the state flag (to indicate which state the task is in, either READY, or SLEEPING, or RUNNING), the process time, the period, list_head for the linked list structure, and a timeval pointer which stores the start time of this process.

Then, we declared the same two global variables as MP1: proc_dir, proc_entry. After that, we created a mutex for the process list, a spin lock for the timer, a cache for new node, a task_struct for the dispatching thread, and a pointer to store current running task.

mp2_read() is very similar to mp1_read(), except that we store pid, period, and process time this time. The userapp will utilize mp2_read() to check whether or not it is registered.

In the function ‘pick_task_to_run’, we find the next task to run. Since the list is sorted in terms of increasing period (decreasing priority), we only need to find the first task in READY state, and make it the next task. Meanwhile, preempt current running task if any. For the new task, we wake it up and give it a high priority. It is also important to note that now it is the current_running_task and its state is changed to RUNNING. The old task is set to READY state.

dispatching_thread() calls pick_task_to_run() to pick next task. This function is called whenever a YIELD message is received, or a wake up timer expires. After it wake up the picked task, it will sleep.

The function ‘_wakeup_timer_handler’ uses a spinlock, because when the handler tries to lock it but failed, the handler will re-try. It is a very useful in code to handle the interrupt handlers. Inside it, we change the state of current node to the READY state and wake up the process. The function ‘read_process_info’ parses the process info and store it in the nodes.

The function ‘init_node’ is where we initialize our variables, setting new task’s to SLEEPING state, initializing its timer, and getting the start time, etc.

The function ‘add_to_list’ adds the node to the list at the appropriate position. In order to do this, we need to traverse through the list and compare the period of each element. The list will be sorted in increasing order of period.

The function ‘destruct_node’ frees up the task node.

‘find_task_node_by_pid’, as its name suggests, finds the task node by its pid. Note that we used mutex lock to make sure only one is being used at a time.

The function ‘yield_handler’ sets the yield tasks to the sleeping mode and starts the wake-up timer. Inside it, we use the ‘find_task_node_by_pid’ function mentioned above.

In the function ‘admission_control’, we use the equation given in the spec. We multiply the process time by 1000, so that the ratio we use will be 693, thus avoiding floating point arithmetic.

In ‘mp2_write’, the beginning part is very similar as ‘mp1_write’, but then we add some things to the function. Firstly, if the ‘buf’ does not pass the admission control, then we immediately return 0. Then, we construct if/if-else/else statements to check the first character of the ‘buf’ and perform the appropriate functions.

While doing writing, we created a new self-defined task node, and insert it into the existing task linked list based on its period (shorter period has higher priority).

‘mp2_init’ is invoked when a new module is loaded. This is also very similar to ‘mp1_init’, with creating a cache, initializing the global variable, and dispatching a task using ‘dispatching_thread’

‘mp2_exit’ also executes similar things as ‘mp1_exit’, clear up all the memory.

In the ‘userapp.c’ file, we added a lot more stuff than ‘userapp.c’ for MP1.

reg() writes the process’s pid, period, and process time to /proc/mp2/status. The ‘unreg()’ function we write the pid of the task that is to be unregistered. The ‘yield()’ function we write the pid of the task to be yielded. The ‘do_job()’ function calculates the Fibonacci number. The ‘check_status’ function reads the PROC_FILE and goes through each line of it. If there is a matching PID, we return 0; otherwise, -1.

In the main function, after initializing the pid and checking it for validity, we run this in a for loop for 5 iterations to test it.

To run the program:
Type ‘make’ in terminal to compile the code.
Next, type ‘sudo insmod mp2.ko’.
Then, type ‘./userapp 1000& ./userapp 500&’ to run ‘userapp’, here we start 2 tasks and 1000 and 500 are their period in millisecond.

Check the output pid, wake-up time and process time

```
cs423@sp16-cs423-g23:~/CS423/MP2$ make
rm -f userapp *~ *.ko *.o *.mod.c Module.symvers modules.order
make -C /lib/modules/3.19.0-25-generic/build M=/home/cs423/CS423/MP2 modules
make[1]: Entering directory `/usr/src/linux-headers-3.19.0-25-generic'
  CC [M]  /home/cs423/CS423/MP2/mp2.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/cs423/CS423/MP2/mp2.mod.o
  LD [M]  /home/cs423/CS423/MP2/mp2.ko
make[1]: Leaving directory `/usr/src/linux-headers-3.19.0-25-generic'
gcc -o userapp userapp.c
cs423@sp16-cs423-g23:~/CS423/MP2$ sudo insmod mp2.ko
cs423@sp16-cs423-g23:~/CS423/MP2$ ./userapp 1000& ./userapp 500&
[1] 8110
[2] 8111
cs423@sp16-cs423-g23:~/CS423/MP2$ pid: 8111, wake-up time: 499 ms
pid: 8111, proc time: 46 ms
pid: 8111, wake-up time: 454 ms
pid: 8111, proc time: 46 ms
pid: 8110, wake-up time: 1045 ms
pid: 8110, proc time: 47 ms
pid: 8111, wake-up time: 454 ms
pid: 8111, proc time: 46 ms
pid: 8111, wake-up time: 454 ms
pid: 8111, proc time: 46 ms
pid: 8110, wake-up time: 955 ms
pid: 8110, proc time: 46 ms
pid: 8111, wake-up time: 454 ms
pid: 8111, proc time: 46 ms
pid: 8110, wake-up time: 954 ms
pid: 8110, proc time: 46 ms
pid: 8110, wake-up time: 954 ms
pid: 8110, proc time: 46 ms
pid: 8110, wake-up time: 954 ms
pid: 8110, proc time: 46 ms

[1]-  Done                    ./userapp 1000
[2]+  Done                    ./userapp 500
cs423@sp16-cs423-g23:~/CS423/MP2$
```

Finally, type ‘cat /proc/mp2/status’ to see if all tasks successfully unregistered themselves.
In order to uninstall module, type ‘sudo rmmod mp2’.