CS 423 MP2 Group 9

Laurynas Tamulevicius tamulev2
Pranava Aditya paditya2
Vassil Mladenov mladeno2
David Lipowicz lipowic2

Design and Implementation:

For our design and implementation, we followed the following steps:

1. To begin this MP, we used our code for the linked list helper functions and also reused and modified some of the generic functions from MP1. This gave us a skeleton to start with in our implementation of MP2.
2. The next step in our design process was to implement the Proc Filesystem Entry as specified in the MP document. In order to do this, we used a switch statement to break the implementation into three distinct cases based on an operation character as suggested in the MP specification. We then use this operation character to switch to one of three functions: REGISTER(pid, period, cputime), YIELD(pid) and DEREGISTRATION(pid),
3. Our next step was to augment the Process Control Block (PCB). In order to do this, we declare an additional data structure named mp2_task_struct. We use the helper function find_task_by_pid() to find the task_struct associated with a given PID.
4. After setting up our PCB struct, we implemented the registration for a task. In the REGISTER(pid, period, cputime) function, we used kmem_cache_alloc in order to allocate a new instance of our mp2_task_struct as this improves performance and reduces memory fragmentation. The task is initialized in the SLEEPING state. We then initialize the other member variables of the struct and add the task to the list using the list_add() function in order to complete the registration process.
5. The next step in our implementation was to implement the mechanism to de-register a process. In the DEREGISTRATION(pid) function, we remove the task from the list using the list_del() function and de-allocate all the data structures that we allocated in the registration step.
6. This step involved us implementing our dispatch_thread which calls the dispatch_thread_function. In this, we call handle_new_running_task which finds the task with the shorted pid and wakes up the process. We also change the state of it to RUNNING along with changing the old task from RUNNING to READY. We then handle the task preemption and context switches using the sched_setsceduler() function.
7. Next, we implemented the wake up timer. The timer wakes up the dispatch_thread and sets the task state to READY.

8. We then implemented the YIELD function. We check if the next period has started yet, and if not, we put the task into a SLEEPING state and calculate the start time of the next period. We then also set the timer and set the task state to TASK_UNINTERRUPTIBLE.
9. This step involved us adding in admission control. We checked whether the new task could be added to our list without it causing any existing tasks to miss their deadlines. In order to properly implement this, we used the formula given in the specification and made sure that we did not have any floating point arithmetic.
10. Our final step when working on the kernel code was cleaning up the code to make it more readable, along with making sure that memory was properly managed everywhere.
11. We finally created our test application, which is a simple factorial function. After that, we tested the program as a whole.

Testing Instructions:

The steps to test our program are as follows:

1. Compile the program using the make command.
2. After the program compiles, run sudo insmod mp2.ko to.
3. After installing the kernel module, run ./userapp <period> <number of jobs>
4. The next step is to run cat /proc/mp2/status and check if the processes are registered properly.
5. After this, run dmesg to see the kernel messages in order to check if reflect what is supposed to happen in the program.