OPERATING SYSTEM PROJECT 1 REPORT

李佳恩 B07902096

Design

System Calls

1. **sys_pj1_print**
   will receive the 3 parameters `pid, start_time, end_time.` `printk` will print the dmesg containing the start time and end time of a specific process
2. **sys_pj1_time**
   This method firstly receives a `struct timespec` pointer then stores the time info by `getnstimeofday` into the pointer in seconds and nanoseconds time measurement.

Driver

The `driver.c` program is the main function that act as the "brain" to drive the whole process. The header files needed declared here. Inside the main function, there are some process that handle the input and then categorized the policy that the inputs want here before forwarding the input data to the next step.

Process

The Process files (process.c and process.h) mainly contains the utility, the variables and Process Handling needed. These functions and data later may be used by the other part of files.

1. **Utility**
   The Struct constructed is to store the process' information, such as the `name, readytime, executiontime and pid.`

   `TIME_UNIT` time unit is a given time to do the million iterations in an empty loop. Before the forked child process starts and after finish running its TIME_UNIT, it will record its starting time from the system call `sys_pj1_time` and print them just before its exit via `sys_pj1_print`

2. **Process Handling**
   `Process.c` file mainly contains the process handler method. `proc_assign_cpu` to assign process to specific core, setting the CPU affinity for given pid. `proc_exec` to Execute the process and return pid, `blockprocess` to set the very low priority to process and the other hand `wakeupprocess` is a function to set the high priority to the process.

Scheduler

`schedpolicy.c` contains the whole process of scheduling policy (FIFO, Round- Robin, Short Job First, and PSJF). Inside this file, the input from the `driver.c` processed, including the sorting process according to the ready time, set the affinity CPU, and the process of the corresponding scheduler. Inside this file there is also the sorting comparison utility that helps the quicksort algorithm to determine the smaller ready time. Below are the explanation for each processes:

1. **FIFO**
   Based on its definition, First In First Out Scheduling will firstly check the process that is ready to be executed at a particular time. Then, it will set its priority to the low mode, and then the ready process will be executed. Whenever FIFO try to look for the next process to run, it will consider 2 conditions: if there is no running process, then the pointer will point to the next ready process in the queue, on the other hands if there is running process, then it will finish the running process first, then continue to search for the next. When all job is done (the pointer is located at n - 1 (n is the number of processes, given in the input) and that job is over), finish this scheduler.

2. **RR**
   The basic structure of RR scheduler is same as FIFO. But we need to add a queue to maintain the order of unfinished but started jobs. The different in the implementation is, when there is a running process but the time quantum has passed (a process that runs for kRRRound (which is 500 by the project definition) continuous time units), then the it changes the ready time to the current time and put it back to the queue. After that, the queue will be examined again to look for the next ready process to be executed.

3. **SJF**
   The basic structure of SJF is also same as FIFO and RR. The processes are stored in order of their running time. When there is no process is running, the first (with smallest running time) will be pop out and start executing.

4. **PSJF**
   The structure is same as SJF, except that when every new process is ready, we have to check whether the execution time is less than the process that is currently running. If yes, then we have to switch these two process (push the running process into the ready queue and run the other shorter execution time).

Kernel Version

Kernel 4.14.25 on Linux Ubuntu16.04

Comparison

All of the output results (`stdout` and `dmesg`) stored inside the output folder. If we compare the result of the program with the result that we can get based on the theoretical theorem of each scheduling process, the finish orders are the same. If we want to compare with more detailed methods, for sure there is a small difference error happens, but it is still quite reasonable because the outputs inside this process are true. The error variation can happen because the scheduler speed can be varying, and it might decrease and impact the processing time to be longer. This happens because our computer must run another process also, not only our program, so that it might slow down the execution.