IMPLEMENTATION OF I/O BASED OPERATIONS

CMPE322 project series is a set of projects that teaches how to construct an operating system, named BU*nix* *(written in institutional colors of Boğaziçi University ☺)*, from scratch. In the first project, you scheduled processes which are independent from each other. In the second project, you synchronized them with binary semaphores. In this project, you will implement a mechanism for your operating system in order to handle the I/O based operations.

Scenario in the Third Project

BUnix manages a hardware which have two printers and one hard-drive. You inform your BUnix coders that if they need to use a printer, they have to write "dispM_[Printer_No]" in which Printer_No is 0 or 1. Also, if they want to read a block from the hard-drive, they have to write "readM_[Block_No] in which Block_No is from 0 to 1000. After all, they compile their new codes which look like in Table 1.

INSTRUCTION NAME	INSTRUCTION EXECUTION TIME (ms)
instruction_1	20
instruction_2	20
dispM_0	120
instruction_3	50
instruction_4	20
dispM_1	150
readM_28	120
instruction_5	30
instruction_6	30
readM_854	100
instruction_7	30
. . .	. . .
exit	10

Table 1 – A Text Based Program Code File Example

CPU Bound vs. I/O Bound

In the previous projects, we always run the CPU-bound processes that need to use the available CPU power to complete their instructions. However, in this project the processes also run the instructions that do not need CPU power but have to wait the peripherals (printers and hard-drive) to complete their tasks. When the processes run these instructions, you have to send these processes to the corresponding wait queues until their tasks are completed by the peripherals.

In addition to your previous code, we want you to implement the following parts in your third project:

- Implement a wait queue for each printer and for the hard-drive (three wait queues in total). These wait queues should be a FIFO again.
- Implement "dispM" operation:
    - You have to send the process to the corresponding wait queue.

- The process at the head of the queue should be sent back to the tail of the ready queue after the “instruction execution time”, which is given in the second column of the dispM command.
- The other processes in the wait queue should wait the processes ahead of them (FIFO Queue).
- When a process reaches to the head of the wait queue, you have to count down the “instruction execution time” before sending back it to the tail of the ready queue.

- Implement “readM” operation:
    - Implementation of this operation is similar with the “dispM”.
    - However, we have a cache in the memory region which can store 2 blocks.
    - At the initial step, the cache area is empty.
    - First you have to check whether the “block_no” is available in the cache area.
    - If so, you should not send the process in the wait queue of hard-drive. The execution time of the “readM” is 0 if it is in the cache. You can run the following instruction of the process.
    - Otherwise, you have to send the process to the wait queue of the hard-drive. Then, operate the queue similar as the “dispM” command. The “instruction execution time” is given in the second column of the “readM” command.
    - Do not forget to update the cache. The replacement in the cache is “Least Recently Used” (LRU) basis. You have to overwrite the new block on the least recently used block.
- An output mechanism that generates a file to show the ready queue. Ready queue output name should be “output.txt” again. You should display the queue only at the “context switch” time. Do not print if the following events is occurred without “context switch”:
    - If a new process is arrived to the queue.
    - If a process is arrived from a wait queue.
- An output mechanism that generates files for the wait queues whenever they are changed by the algorithm.
- Wait queue output names should be:
    - First printer (DispM_0): “output_10.txt”
    - Second printer (DispM_1): “output_11.txt”
    - Hard drive (ReadM_X): “output_12.txt”
- The format of the files should be the same as the previous projects.

Notes

- In our test definition files:
    - P1 will always arrive at time 0
    - Process arrival times will increase in each row.
    - There will be always a process in the ready queue according to test definition files. So you don't have to worry about what to do in "idle CPU" state.
- If more than one process needs to add the ready queue at the same time, the priority is:
    - The process from the wait of the First printer
    - The process from the wait of the Second Printer
    - The process from the wait of the Hard-Drive
    - New Arrived Process

    - The Previous Running Process
- Do not forget that the instructions are atomic operations! You could not change the queues while you are running an instruction. For example:
    - Assume that P2 runs “dispM_0 40” at 280. There is not any process in the wait queue of printer 0. Hence, the printing operation will finish at 320.
    - P1 starts to run “instr3 70” at 280. The operation will finish at 350. Therefore, you have to dequeue P2 from the queue of dispM_0 and send back it to the ready queue at 350.
- If you could not complete the second project, you don’t have to implement it in this project. We will not test your code file with “waitS” and “signS” commands.

Development Platform

You have to implement your design in the Linux Platform with GCC/G++ Compiler. We strongly advise you to install Ubuntu which has pre-installed GCC/G++ compiling tools. We will test your code in this platform and if your compiled code is not compatible with the test platform, you might be invited for a demo, if necessary.

Provided Files

The following files are given together with the project:

- The process definition file (number of processes and their arrival times will be modified to test your code).
- Four program code files (number of instructions, their names and their execution times will be modified to test your code. Only exception is the last instruction name is always “exit” and its execution time is 10 milliseconds).
- The expected output file.

Project Requirements

- Your project should have generated the expected output file for modified process definition file and program code files. (80%)
- The source code documentation. The documentation depends on your design style but at least having comments that explain your code is strongly advised. (20%)