CSE 325 - Project 3

Project Overview

In this assignment, you will design and implement a C++ program that simulates the behavior of a **5-state processing model**. You will complete the exercises below and submit your work for grading via the D2L system.

Project Deliverable

The deliverable for this assignment is the following file:

proj03.cpp – your program code

Project Specifications

- You will simulate a 5-state process model in which the OS kernel handles the execution of a set of processes using a single CPU core. You will need to create data structures representing the Ready, Blocked, and Running states and simulate the execution of instructions for the process that is running in each clock cycle.
- The processes will be provided as input files in the same directory as the program's executable. The file format is as follows:
    1. The name of each process file will be *processN*, where *N* is a number between 1 and 9. For example, if there are two input processes, the files will be *process1* and *process2*. (Note: These files have no extensions but can be opened with any text editor.)
    2. The first line of each *processN* file will contain a single integer, representing the initial priority of the process (higher values = higher priority). For this simulation, priority is only relevant at the start and has no impact in an ongoing run.
    3. The remaining lines in the file will contain instructions that should be executed by the CPU. You should simulate a time cycle system where, in each cycle, one instruction from a process is executed (CPU idle time should be minimized). There is no need to simulate registers or the actual outcomes of the instructions.
- Some instructions will be system calls. Since handling most system calls takes time, a process issuing a system call should be blocked until the interrupt is handled. The format of system call instructions differs from regular instructions, and if applicable, they will specify the number of clock cycles required to handle the interrupt. Consider the following system call example:

```
SYS_CALL, NETWORK 3
```

In this example, a network-related system call is issued, causing the process to be blocked for 3 CPU cycles. Upon executing this instruction, the process moves to the Blocked state for 3 cycles. Once the blocking period ends, it transitions to the Ready state, where it waits for the kernel to schedule it for execution.

All system call instructions begin with the keyword SYS_CALL. The second part of the instruction will be one of the following:

  - TYPE N – Where TYPE represents the system call type, and the integer N specifies the number of cycles the process will be blocked.
  - TERMINATE – Indicates that the process requests termination (usually the last line of code).

- When you run your program, it should read all *processN* files and move them to the Ready Queue, sorted by initial priority. The kernel should then select the highest-priority process and move it to the Running state.
  - Each clock cycle, the running process executes one instruction.
  - If the instruction is a system call that blocks the process, the process should be moved from Running to Blocked, and a new process should be selected to run.
  - **At the end of each clock cycle**, the program should check whether any blocked processes are ready to return to the Ready Queue after their blocking time expires.
- You should simulate a timer interrupt with an interval of **5 clock cycles**. When a timer interrupt occurs, the Running process should be moved to the end of the Ready Queue (preemption), and a new process should be selected to run. If a new process enters the Running state, the timer should reset.
- Your program should log execution events from start to finish in a file called LOG.txt. This is the output of your program. The log should be formatted properly and easily readable and include a cycle counter (check test case outputs). The following events should be logged:
  - All state Transitions. For example:

```
Process 1: Ready -> Running
Process 1: Running -> Blocked
Process 1: Running -> Ready
Process 1: Blocked -> Ready
Process 1: Running -> Halted
```

Project Notes

1. As stated above, your source code file will be named "proj03.cpp" and you must use "g++" to translate your source code file in the *scully* Linux environment.
2. The following is a valid execution of the program:

```
./proj03 N
```

   Where N indicates the number of the input process files. For example the following command indicates that there will be 3 process files:

```
./proj03 3
```

3. if *-debug* is provided, your program should also log each executed instruction. Example command:

   ```
   ./proj03 4 -debug
   ```

4. This is a simulated system with its own rules. It does not represent the behavior of modern operating systems.
5. The process ID (PID) is determined by the N value in the process file name. (For example, 'process1' has PID = 1.)
6. In each cycle, it it's possible, an instruction should be executed.
7. If a process has no remaining instructions, it should be terminated.
8. Please follow the formatting in the test cases that are shared with this example. While slight formatting mismatch such as the ones caused by whitespace and case-sensitivity are not important, you must attempt to keep the formatting and the text as similar as possible to the test cases. **Do not print any additional text to the LOG.txt file.**
9. When returning a process to the ready queue, you should always add it to the back of the queue (last place).
10. You may assume that the process files are formatted correctly and only contain the priority integer, normal instructions and system calls. Still, you should check if the *processN* files exist or not.