CS 3113 – Intro to Operating Systems – Spring 2025
Project Two: Due February 28, 20205, 11:59 PM

Description: This project is an extension of Project One with following enhancements and modifications have been introduced to improve the simulation of process management:

1. CPU Time Allocation and Timeouts
   - A new parameter, CPUAllocated, specifies the number of CPU ticks a process can execute before it times out.
   - If a process runs for more than CPUAllocated ticks without terminating or issuing an I/O operation (e.g., a print statement), it will be interrupted.
   - The process is then moved to the back of the ReadyQueue, and a message is printed indicating a TimeOUT Interrupt.

2. Context Switch Handling
   - A global CPU clock starts at 0 and increments based on:
     - The number of CPU ticks consumed by an instruction.
     - A new context switch time that accounts for the overhead of switching processes.
     - The CPU clock also increments by the context switch time when the ReadyQueue is empty.

3. I/O Interrupts and I/O Waiting Queue

   - When a process executes a print instruction, it issues an IOInterrupt and moves to a new queue called IOWaitingQueue.
   - The job status changes to IOWaiting.
   - The CPU clock time when the job was moved to the IOWaitingQueue is recorded.
   - Jobs are moved from IOWaitingQueue back to the ReadyQueue once they have spent enough time in the I/O queue, determined by the print instruction's required CPU time.
   - The IOWaitingQueue is checked for job transfers every time an interrupt occurs in the CPU.

4. Process Execution Tracking

   - Detailed output is required whenever a job moves from one queue to another, including:
     - ReadyQueue to Running
     - Running to IOWaitingQueue (for I/O operations)
     - Running to ReadyQueue (on timeout)
     - IOWaitingQueue to ReadyQueue (after I/O completion)

5. Process Termination Logging

   - When a process terminates, log:
     - The CPU clock time when it first entered the Running state.
     - The CPU clock time when it reached the Terminated state.
     - The difference between these two times to track total execution time.

6. Total CPU Time Calculation
   - At the end of execution, print the total CPU time consumed by all processes.

Input Description: The program reads the input file via redirection, and the file must adhere to a specific format. The first line specifies the maximum size of main memory in integers. The second

line indicates the total number of processes. Each subsequent line contains the details for an individual process. Each process description begins with a processID, which serves as a unique identifier for the process, followed by its initial state, which is always set to NEW. Next is the maxMemoryNeeded, representing the memory required for the process, and numInstructions, the number of instructions associated with the process. The instructions are encoded as integers. The supported instruction types are: Compute (encoded as 1 iterations cycles, e.g., 1 5 10), Print (encoded as 2 cycles, e.g., 2 3), Store (encoded as 3 value address, e.g., 3 42 250), and Load (encoded as 4 address, e.g., 4 250). Each instruction type specifies the operation and its associated parameters. This structured input format enables the program to parse and manage process information effectively.

Example Input and explanation:

```
2048 5 2
2
1 300 3 1 5 10 3 42 250 4 250
2 400 4 2 3 3 100 350 1 2 7 2 5
```

The above input indicates that the main memory has 2048 elements. The number 5 is the total time a process can be in the CPU before it gives a TimeOUT interrupt. The number 2 after that is the time it takes to context switch that is move a job from readyQueue to running state. This CPU global clock is incremented even if there are no jobs in the readyQueue. On the next line, the value 2 represents two process that needs to be executed. The instructions to be executed will be compute, store, load, and print. These will be discussed below. **Store and Load operation take one CPU Time.**

Process 1 has a processID of 1 and an initial state of NEW. It requires a maximum of 300 units of memory (int array of size 300) and includes 3 instructions. The instructions are as follows: 1 5 10, which represents a Compute operation with 5 iterations consuming 10 CPU cycles; 3 42 250, which represents a Store operation that saves the value 42 at address 250; and 4 250, which represents a Load operation that retrieves a value from address 250.

Process 2 has a processID of 2 and an initial state of NEW. It requires a maximum of 400 units of memory (int array of size 400) and includes 4 instructions. The instructions are as follows: 2 3, which represents a Print operation consuming 3 CPU cycles; 3 100 350, which represents a Store operation that saves the value 100 at address 350; 1 2 7, which represents a Compute operation with 2 iterations consuming 7 CPU cycles; and 2 5, which represents a Print operation consuming 5 CPU cycles.

Output Requirements: In addition to the output requirements for Project One, we need the following for this project, modify the given main program to include the following additional outputs:

- Print a message whenever a process is moved between queues.
- When a process times out, print:

  `Process <ProcessID> has a TimeOUT interrupt and is moved to the ReadyQueue.`

- When an I/O interrupt occurs, print:

  ```
  Process <ProcessID> issued an IOInterrupt and moved to
  IOWaitingQueue.
  ```

- When a process moves from **IOWaitingQueue** to **ReadyQueue**, print:

  ```
  Process <ProcessID> completed I/O and is moved to
  ReadyQueue.
  ```

- When a process terminates, print:

  ```
  Process <ProcessID> terminated. Entered running state at:
  <StartTime>. Terminated at: <EndTime>. Total Execution
  Time: <Duration>.
  ```

- At the end of execution, print:

  ```
  Total CPU time used: <TotalTime>.
  ```

Rules and Submission Policy

All projects in this course are **individual assignments** and are not to be completed as group work. The use of **automatic plagiarism detection tools** will be employed to ensure academic integrity. Collaboration with others or the use of outside third parties to complete this project is strictly prohibited. Submissions must be made through **GradeScope**, where automated grading will be conducted. Additionally, manual grading may be performed to ensure correctness and adherence to requirements.

Several input files will be used to evaluate your program. While a subset of these input files will be provided to you for testing, additional files not shared beforehand will also be used during grading. Your score on this project will depend on producing correct results for all input files, including the undisclosed ones used in GradeScope evaluation.

All programs must be written in **C++** and must compile successfully using the **GNU C++ compiler**. It is your responsibility to ensure that your program adheres to these requirements.

The course syllabus provides more details on the late and submission policy. You should also go through that.