CS 3113 – Intro to Operating Systems – Spring 2025
Project Two: Due March 16, 20205, 11:59 PM

Description: This project extends Project Two by introducing a new queue called NewJobQueue, which manages job loading into ReadyQueue while considering memory constraints. The goal is to simulate a memory management system where jobs are dynamically loaded into available memory spaces and removed when they complete execution.

Enhancements Over Project Two

1. Introduction of NewJobQueue
    - Jobs first arrive in NewJobQueue instead of being directly placed in ReadyQueue.
    - Jobs are moved from NewJobQueue to ReadyQueue only if there is sufficient available memory in the system.
    - Jobs remain in NewJobQueue until enough contiguous space is available.
2. Dynamic Memory Allocation
    - Memory is tracked using a linked list, where each node represents a memory block with the following fields:
        - ProcessID (Integer): ID of the process occupying the block (-1 if free).
        - Starting Address (Integer): The memory location where the block begins.
        - Size (Integer): The size of the memory block.
    - Initially, the memory starts as a single large free block.
    - Whenever a job is moved from NewJobQueue to ReadyQueue, the linked list is searched for a sufficiently large free block.
    - If no free block is large enough, the job remains in NewJobQueue until sufficient memory is freed.
3. Handling Job Termination
    - When a job terminates, the memory it occupied is marked as free.
    - New jobs are loaded into memory using available free space.
4. Memory Coalescing (Merging Free Blocks) When Needed
    - Memory coalescing is triggered only when a job cannot be moved from NewJobQueue to ReadyQueue due to insufficient contiguous memory.
    - When triggered, the linked list is scanned to merge adjacent free blocks into larger free spaces.
    - After coalescing, the NewJobQueue is rechecked to see if any waiting job can now be moved into memory.

Output Requirements: In addition to the outputs required in Project Two, additional logs are needed:

- When a job moves from NewJobQueue to ReadyQueue, print:
  `Process <ProcessID> loaded into memory at address <StartingAddress> with size <Size>.`
- When a job cannot be moved due to insufficient memory, print:
  `Process <ProcessID> waiting in NewJobQueue due to insufficient memory.`
- When memory coalescing is triggered, print:
  `Insufficient memory for Process <ProcessID>. Attempting memory coalescing.`
- After memory coalescing, if space is available, print:
  `Memory coalesced. Process <ProcessID> can now be loaded.`
- When a job terminates and releases memory, print:
  `Process <ProcessID> terminated and released memory from <StartingAddress> to <EndingAddress>.`

Execution Flow

1. Initialize the memory as a single large free block.
2. Process jobs in NewJobQueue, searching for available memory.
3. Load jobs into memory and move them to ReadyQueue when space is available.
4. Execute jobs according to the scheduling policy from Project Two.
5. Handle job termination and update memory.
6. If a job in NewJobQueue cannot be moved due to insufficient memory, trigger memory coalescing.
7. Recheck NewJobQueue and attempt to load jobs after coalescing.
8. Continue until all jobs are executed.

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.