**INCS 102 Operating Systems,** Winter 2024

**Project Description Due Date: 06/12/2024 at 11:59pm**

**Project Overview:**

This project aims to implement a multi-core execution system with a master-slave architecture, where one core acts as the master coordinating execution, with two cores serve as slave workers processing tasks concurrently. **The project will be implemented using Java**.

**Input:**

You will be provided with a file, containing programs with instructions for variable assignments, arithmetic operations, and print commands. Although different program files may be used in evaluations, they will adhere to the same format and syntax.

**Assumptions:**

* All programs arrive at time zero.

**Components to Implement:**

* **Ready Queue**: A queue to manage processes waiting for execution, allowing the master core to assign tasks to slave cores.
* **Memory**: Shared memory for storing variables and data accessible by all cores, ensuring data consistency across the master and slaves.
* **Process Control Block (PCB)**: Each process will maintain a PCB that includes the Process ID, Program Counter, and Memory Boundaries.
* **Master-Slave Architecture**: The master core will schedule and delegate tasks to the slave cores, coordinating execution and resource management.

**Scheduling Algorithms:**

* **Round Robin:** Implement a Round Robin scheduling algorithm that allocates time slices of 2 **(quantum=2)** to each process on a rotating basis across the available cores.

**Functions/Operations to Implement:**

* **Variable Assignment**: Assign values to variables as specified in the programs, ensuring synchronization between cores.
* **Arithmetic Operations**: Support addition, subtraction, multiplication, and division operations across multiple cores.
* **Print Commands**: Handle output display from each core in a thread-safe manner, ensuring accurate representation of program results.

**Execution Monitoring:**

* Implement monitoring that tracks and prints the currently executing process on each core. The master core will provide updates on the completion of processes, detailing when each process finishes at each cycle.

### **Implementation Steps:**

#### 1. Initialize Queues and Memory:

* **Create a Data Structure for the Ready Queue**.
* **Implement Memory Allocation**:
  + Design a shared memory management system to store variables and program data, accessible by all cores.

#### 2. Parsing Programs:

* **Read Input Text Files**.
* **Parse Instructions**.
* **Categorize Instructions**:
  + Identify and classify instructions into variable assignments, arithmetic operations, and print commands.

#### 3. Implement Operations:

* **Variable Assignment**:
  + Implement the 'assign' operation to handle both direct value assignments and user input.
* **Arithmetic Operations**:
  + Implement addition, subtraction, multiplication, and division operations that can be executed by slave cores.

#### 4. Scheduling Algorithms:

* **Implement Scheduling Algorithms**:
  + Develop Shortest Job First (SJF) algorithm to manage the execution of processes across cores.
* **Define Parameters**:
  + Establish criteria for task prioritization in Shortest Job First.

#### 5. Execution Control:

* **Iterate through Parsed Instructions**:
  + The master core will delegate tasks to slave cores based on the selected scheduling algorithm, managing execution flow.
* **Print Current Process**:
  + Output the currently executing process on each core at each clock cycle.
* **Track Process Completion**:
  + Maintain logs to track when each process finishes execution, updating the system status in real-time.

#### 6. Testing and Debugging:

* **Test the implementation with various input programs to ensure correct execution.**
* **Identify and resolve any bugs encountered during testing, ensuring that the system operates as intended.**

**Hint: You need to extend the Thread class in your multi-core implementation using the keyword** extends Thread.

### **Expected Outputs at Each Clock Cycle:**

1. **Ready Queue**:
   * Display the contents of the Ready queue after each cycle, reflecting the processes awaiting execution.
2. **Processor Status**:
   * Indicate which process is currently executing in each slave core.
3. **Memory State**:
   * Showcase the state of memory after each operation by Listing the variables and their assigned values.

### **Evaluation Criteria:**

1. **Correctness**:
   * Verify that each implemented operation functions correctly.
   * Ensure that variables hold correct values after assignments and arithmetic operations.
2. **Scheduling Algorithms**:
   * Assess the performance of Round Robin algorithm by checking if Round Robin correctly rotates processes based on time slices.
3. **Execution Monitoring**:
   * Validate that the program accurately displays the current executing process.
   * Ensure that process completion tracking is precise for each clock cycle.
4. **Memory Management**:
   * Verify the efficiency of memory allocation and deallocation.
   * Check for memory leaks or inefficient memory utilization.
5. **Error Handling and Robustness**:
   * Test with various input scenarios, including edge cases, to ensure robust error handling.
   * Confirm that the program handles unexpected inputs gracefully without crashing or hanging.
6. **Performance**:
   * Assess overall performance in terms of execution time across different programs.
   * Measure the efficiency of scheduling algorithm in managing concurrent processes.
7. **Debugging**:
   * Identify and resolve any issues or bugs encountered during testing.
   * Ensure that the program is free from logical errors and runs smoothly under various scenarios.

**Submission:**

* You are required to submit the src file with the programs all compressed in **.zip format** and **named after the team number name.**
* The deadline is the **06 December 2024 at 11:59 pm** to this email address: os.incs102@gmail.com.
* **Any other format of submission will be considered as zero!**

***Goodluck!***