BIRZEIT UNIVERSITY
Electrical and Computer Engineering Department

ENCS339 Operating Systems Spring 2022/2023

OS Projects

Project Instructions

Project Scope: Each project aims to develop an operating system simulator that covers one of the following topics: Threads, Process Synchronization, CPU Scheduling, and Main Memory Management.

Group Formation: Form groups of **three students each**. Each group will work together on the project, sharing responsibilities and collaborating on the implementation. (Each group must choose one of the topics: *Threads, Process Synchronization, CPU Scheduling, and Main Memory Management*.)

Project Timeline: You must present a progress report every 10 days. The progress reports are scheduled as on: June 1st, June 11th , June 21st , July 2nd The **DEADLINE** for the final project submission will be **July 6th 2023**.

Project Deliverables: Each group is responsible for the following deliverables:

- Source code: Develop the operating system simulator from scratch, ensuring originality and avoiding plagiarism.
- Documentation: Prepare detailed documentation that explains the design and implementation details.
- Presentation: Create a presentation to showcase the simulator's features, discussing the design choices, implementation challenges, and overall functionality of the system.

Genuine Work: Each group must ensure that their work is original and genuine. It is strictly prohibited to copy code or solutions from external sources, including online platforms or previous projects. You are expected to utilize your own understanding, creativity, and problem-solving skills to tackle the project requirements.

Collaboration and Division of Tasks: It is very expected that group members will work together, share knowledge, and provide support to ensure a cohesive and successful implementation. However, ensure that each member of you has a fair share of responsibilities and actively contributes to the project. You might follow these guidelines:

1. *Task Identification: Start by identifying the key tasks required for the project. Break down the project into smaller components or modules, and make each component assigned to one or more group members.*

2. *Group Discussion: Conduct a group discussion to share information about the project requirements, the scope of each task, and the expected deliverables. Group members should ask questions, clarify doubts, and share their insights and ideas.*

3. *Task Assignment: Distribute the tasks among group members based on their interests, strengths, and availability. Consider assigning tasks that align with each member's expertise or areas they want to explore. Each group member should has at least one significant task to contribute.*

4. *Collaboration Channels: Set up collaboration channels for effective communication and knowledge sharing within the group. This could include group meetings, online chat platforms, shared document repositories, or project management tools. Please actively participate in discussions, share their progress, and seek help when needed.*

5. *Regular Updates: Establish regular checkpoints or milestones where each group member provides updates on their assigned tasks. This could be through progress reports, demonstrations, or code reviews. Regular updates allow group members to stay informed about the project's progress and provide feedback or assistance if required.*

6. *Peer Learning and Support: Group members should support each other by sharing their knowledge, experiences, and resources. You are welcome to seek help, or assist your peers.*

7. *Task Dependencies: Identify any dependencies between tasks and ensure effective coordination among the members. It is very important to communicate with each other's regarding task dependencies (I can provide support in resolving any conflicts or challenges that arise due to dependencies).*

8. *Group Meetings: Schedule regular group meetings to discuss the overall project status, address any issues or roadblocks, and ensure that all group members are on the same page. These meetings provide an opportunity to synchronize efforts, address concerns, and maintain project momentum.*

9. *Fair Contribution: Accountability and open communication within the group are ultimately very important. If any imbalances arise, facilitate discussions among yourselves to resolve them and redistribute tasks if necessary.*

Code Documentation and Best Practices: It is very important to properly document your the code. Each group must include comments, meaningful variable names, and maintain a clear code structure. Use coding best practices to enhance code readability, modularity, and reusability.

Testing and Validation: It is very important to perform testing and validation of the operating system simulator. Each group should develop comprehensive test cases that cover different scenarios and edge cases.

Academic Integrity: Plagiarism, including copying code or solutions from other sources without proper attribution, is strictly prohibited. It is very crucial to uphold ethical standards throughout the project.

Project Presentation and Evaluation: Project discussion and presentation will be on July 10^{th} (for section 1) and July 12^{th} (section 2)

Each group will showcase their operating system simulator and explain its functionalities, design choices, and implementation details. The evaluation will be based on the quality of the implementation, adherence to project requirements, documentation, presentation skills, and individual contributions.

Project 1

Thread Management Simulator

Implement a thread management system that allows the creation, execution, and synchronization of multiple threads. Include functionalities such as thread creation, termination, synchronization primitives (e.g., locks, semaphores), and context switching.

In the Thread Management component of the operating system simulator, you will implement a system that handles the creation, execution, and synchronization of multiple threads. Here are some additional details on the functionalities to include:

1. Thread Creation and Termination:
    - Provide an interface or command to create new threads within the simulator.
    - Allow users to specify the entry point function for each thread.
    - Implement thread termination mechanisms, such as explicit termination requests or thread completion detection.
2. Thread Execution and Context Switching:
    - Implement a scheduler that determines the order in which threads are executed.
    - Simulate the execution of threads by executing their associated functions or code blocks.
    - Implement context switching to switch between threads, allowing each thread to make progress and share the CPU.
3. Synchronization Primitives:
    - Include common synchronization primitives like locks, semaphores, or condition variables.
    - Allow users to create and use these primitives to synchronize access to shared resources among threads.
    - Implement appropriate mechanisms to handle blocking and waking up threads waiting on synchronization primitives.
4. Thread Synchronization:
    - Implement thread synchronization mechanisms to ensure proper coordination and order of execution.

- Support operations like thread signaling, waiting, and notifying using synchronization primitives.
- Provide mechanisms for thread coordination, such as thread barriers or thread join operations.

5. Thread Communication:
    - Enable communication and data sharing between threads by implementing mechanisms like shared memory or message passing.
    - Allow threads to exchange information or synchronize their activities using appropriate communication channels.
6. Error Handling and Exception Handling:
    - Implement error handling mechanisms to handle exceptions and errors that occur during thread execution.
    - Provide appropriate error reporting and recovery mechanisms to maintain the stability of the system.

Remember to document and demonstrate the usage of each functionality within the simulator. Additionally, you can provide options to visualize the execution and behavior of threads to enhance understanding and debugging.

Project 2

Process Synchronization Simulator

Develop mechanisms for process synchronization, including shared resource access, mutual exclusion, and inter-process communication. Implement synchronization techniques like locks, condition variables, and message passing.

In the Process Synchronization component of the operating system simulator, you will develop mechanisms to ensure synchronization between processes, including shared resource access, mutual exclusion, and inter-process communication. Here are some additional details on the functionalities to include:

1. Shared Resource Access:
    - Implement mechanisms to manage shared resources among processes.
    - Allow processes to request and release access to shared resources.
    - Ensure that only one process can access a shared resource at a time to prevent data inconsistencies or race conditions.

2. Mutual Exclusion:
    - Implement mutual exclusion mechanisms to ensure that critical sections of code are executed by only one process at a time.
    - Include synchronization techniques such as locks or semaphores to enforce mutual exclusion.
    - Allow processes to acquire and release locks or semaphores to control access to critical resources.
3. Inter-Process Communication:
    - Provide mechanisms for processes to communicate and exchange information with each other.
    - Implement techniques such as message passing or shared memory for inter-process communication.
    - Enable processes to send messages or share data structures to facilitate coordination and collaboration.
4. Synchronization Techniques:
    - Implement synchronization primitives such as locks, condition variables, or monitors.
    - Allow processes to use these primitives to coordinate their activities and access shared resources.
    - Support operations like signaling, waiting, and notifying using synchronization primitives.
5. Deadlock Prevention and Handling:
    - Incorporate mechanisms to prevent and handle deadlocks among processes.
    - Implement deadlock detection algorithms or techniques like resource allocation graphs or bankers' algorithm.
    - Include deadlock resolution strategies, such as resource preemption or process termination, to resolve deadlocks when detected.
6. Error Handling:
    - Implement error handling mechanisms to handle exceptions or errors that occur during process synchronization.

- Provide appropriate error reporting and recovery mechanisms to maintain the stability of the system.

Ensure that the simulator provides a user-friendly interface or commands to demonstrate the process synchronization functionalities. You can include options to visualize the synchronization activities and interactions between processes to aid in understanding and debugging. Document and explain the usage of each synchronization mechanism and provide examples to showcase their effectiveness.

Project 3

CPU Scheduling Simulator

Design and implement various CPU scheduling algorithms, such as Round Robin, Priority Scheduling, or Shortest Job First (SJF). Allow users to configure scheduling parameters, simulate task arrival and completion, and evaluate scheduling performance metrics (e.g., turnaround time, waiting time).

In the CPU Scheduling component of the operating system simulator, you will design and implement various CPU scheduling algorithms. Here are more details on the functionalities to include:

1. Scheduling Algorithm Implementation:
    - Implement common CPU scheduling algorithms, such as Round Robin, Priority Scheduling, Shortest Job First (SJF), or any other scheduling algorithms of interest.
    - Each scheduling algorithm should determine the order in which processes or threads are selected for execution.
2. Scheduling Parameters Configuration:
    - Allow users to configure scheduling parameters, such as time quantum for Round Robin, priority levels for Priority Scheduling, or burst time for SJF.
    - Provide an interface or commands to set and adjust these parameters dynamically.
3. Task Arrival and Completion Simulation:
    - Simulate the arrival of tasks or processes into the system.

- Provide a mechanism to generate a sequence of tasks or processes with their associated arrival times and burst times.
- Simulate the completion of tasks based on their burst times or execution time.

4. Performance Metrics Evaluation:
    - Calculate and evaluate performance metrics to measure the effectiveness of the scheduling algorithms.
    - Include metrics such as turnaround time, waiting time, response time, or CPU utilization.
    - Update and track these metrics as tasks or processes are executed and completed.
5. Visualization and Reporting:
    - Provide visualizations to showcase the scheduling activities, such as Gantt charts representing the execution timeline of tasks or processes.
    - Generate reports or summaries that present the performance metrics for each scheduling algorithm.
    - Allow users to compare the performance of different scheduling algorithms based on the metrics obtained.
6. Dynamic Scheduling:
    - Support dynamic scheduling scenarios, where tasks or processes arrive and leave the system during runtime.
    - Implement mechanisms to handle task arrivals, departures, and context switching between processes or threads.
7. Error Handling:
    - Implement error handling mechanisms to handle exceptions or errors that occur during CPU scheduling.
    - Provide appropriate error reporting and recovery mechanisms to maintain the stability of the system.

The operating system simulator should offer a user-friendly interface or commands to configure scheduling parameters, simulate task arrival and completion, and visualize the scheduling activities. It should also provide options to generate reports or summaries for

performance evaluation. Document and explain the usage of each scheduling algorithm and provide examples to showcase their impact on performance metrics.

Project 4

Main Memory Management Simulator

Implement memory management mechanisms, including virtual memory, paging, and page replacement algorithms (e.g., LRU, FIFO). Simulate memory allocation, deallocation, and page table management to provide an understanding of memory management strategies.

In the Main Memory Management component of the operating system simulator, you will implement memory management mechanisms to handle main memory, including virtual memory, paging, and page replacement algorithms. Here are more details on the functionalities to include:

1. Memory Management Unit (MMU):
    - Implement a Memory Management Unit that handles the mapping between virtual addresses and physical addresses.
    - Simulate the translation of virtual addresses to physical addresses using page tables or page directories.
2. Virtual Memory:
    - Implement virtual memory mechanisms to provide an abstraction of a larger address space than the physical memory.
    - Allow processes to allocate and access memory using virtual addresses.
    - Simulate virtual-to-physical address translation using page tables or hierarchical page tables.
3. Paging:
    - Divide the physical memory into fixed-size pages and map them to corresponding virtual memory pages.
    - Implement page table data structures to track the mapping between virtual pages and physical frames.

- Simulate page table lookups during address translation.

4. Page Allocation and Deallocation:
    - Implement memory allocation and deallocation mechanisms for both virtual and physical memory.
    - Allow processes to request memory allocation and release memory when no longer needed.
    - Maintain data structures to track the allocation status of pages and frames.
5. Page Replacement Algorithms:
    - Implement page replacement algorithms such as Least Recently Used (LRU), First-In-First-Out (FIFO), or any other desired algorithm.
    - Simulate the selection of pages to be evicted from memory when a page fault occurs.
    - Track page replacement decisions and update the page table accordingly.
6. Performance Metrics:
    - Calculate and evaluate performance metrics related to memory management, such as page fault rate, hit rate, or memory utilization.
    - Update and track these metrics as memory allocation, deallocation, and page replacement occur.
7. Visualization and Reporting:
    - Provide visualizations to represent the memory management activities, such as page tables, memory maps, or allocation diagrams.
    - Generate reports or summaries that present the performance metrics for memory management.
    - Allow users to compare the performance of different page replacement algorithms based on the obtained metrics.
8. Error Handling:
    - Implement error handling mechanisms to handle exceptions or errors that occur during memory management.
    - Provide appropriate error reporting and recovery mechanisms to maintain the stability of the system.

The operating system simulator should provide a user-friendly interface or commands to simulate memory allocation, deallocation, and page table management. It should also offer options to visualize the memory management activities and generate reports or summaries for performance evaluation. Document and explain the usage of each memory management mechanism and provide examples to showcase their impact on performance metrics.