AIR UNIVERSITY ISLAMABAD CAMPUS
OPERATING SYSTEMS LAB (CS 225 L) – FALL 2024
CLO-02 GA-04
PROJECT PHASE 1

Due Date: (December 06, 2024, 11:59 PM)

Weightage: 5.625% in theory and 15% in lab

Marks: 400

Instructions:

1. This is a group project having a group size of 3 members.
2. Cross-section groups are **NOT** allowed in any case.
3. The code must be written in C/C++.
4. Submit a single zip file containing all your cpp/c files along with the pdf file of your report.
5. ***Your submission must be a single zip file. No rar or any other format is allowed. The file must be renamed with the roll numbers of your group members' section. For example, if your group member's roll numbers are 22i-1234, 22i-1235, 22i-1236 and your section is B, so, the file must be named as 22i-1234_22i-1235_22i-1236_B. Only one submission per group is required and that must be done by the group representative of that group.***
6. **If your code does not run on our machine while giving demo, you will be awarded a straight ZERO in the project.**
7. **Plagiarism in the project will lead to ZERO marks in the project with referring the case to the Disciplinary Committee**.
8. All submissions must be done on Google Classroom. Any submission outside Google Classroom will not be entertained, whatever the reason is.
9. If you have made any assumptions clearly mention that in the pdf document of your report.
10. **Failing to follow any of the above instruction will result in ZERO marks in the project.**
11. **Start your work today as deadline is firm and will not be extended in any case.**

Question 1 Custom Shell in C/C++

Construct a shell-like program in C/C++ that emulates the functionality of a command-line interface, showcasing fundamental operating system concepts such as **process creation**, **process control** and **inter-process communication (IPC)**. The shell must present a custom prompt allowing users to input commands, which are then parsed and executed through appropriate process management mechanisms. The shell should operate as a persistent **interactive command processor**, remaining active until explicitly terminated by the user through a predefined termination command (e.g., quit).

Functional Requirements:

The shell must handle the following functionalities:

1. **Command Execution**:
   - The shell must support the execution of both **standalone commands** and **commands with parameters**, leveraging system calls for **process instantiation** and **context switching**. Examples include:
     - **Commands without arguments**, such as: ls, date etc
     - **Commands with arguments**, including: ls -a, cp sourcefile destinationfile, kill -92234, ls -l ~/tmp
   - The shell must accurately parse and tokenize user input into **system-level executable formats**, ensuring proper distinction between command identifiers and associated parameters.

2. **Process Creation and Control**:
   - Utilize **process spawning mechanisms**, to create child processes for command execution, ensuring isolation and **independent process contexts** for concurrent operations.
   - The child process should invoke **overlay semantics** using system calls to replace its memory space with the executable program specified by the user.
   - Implement **synchronization primitives** in the parent process to achieve orderly execution of child processes, ensuring the shell adheres to a sequential execution model.

3. **Lifecycle Management and Termination**:

   - The shell must function as a long-running service, continuously accepting and interpreting commands in a cyclic manner until the user provides an explicit termination command (e.g., quit). Upon receiving this command, the shell must execute a controlled termination sequence, releasing any allocated resources and exiting gracefully.

Advanced Functional Requirements:

Enhance the shell's capabilities by implementing a support for **pipelined execution** of commands, leveraging advanced **IPC mechanisms.** The shell should be capable of handling both single and multi-level pipelined command execution. Examples include:

- **Commands with single pipeline**, such as: ls -l | wc
- **Commands with double pipeline** such as: ls | sort | wc

Technical Specifications for Piping:

- Use **unidirectional IPC channels** established via the **pipe()** system call to enable data transfer between processes in the pipeline.
- Perform **file descriptor manipulation** using **dup2()** to redirect standard input and output streams, ensuring proper data flow between pipeline stages.
- Integrate a **command parser** to identify pipeline operators and configure the appropriate process topology for multi-stage execution.

The program must exhibit robust behavior, handling erroneous inputs and unsupported commands gracefully while demonstrating clear operational semantics rooted in core operating system principles. All implementations should be accompanied by detailed internal documentation explaining the use of system calls and their relevance to process management and IPC. Also, write a short report describing how dup and dup2() system calls work by using figures and explaining them.

Question 2 Simulation of Construction Site Management System

Objective:

Design and implement a simulated construction site management system using operating system concepts. This project will focus on managing resources, coordinating tasks among workers, and handling synchronization in a construction environment.

Scenario:

You are tasked with simulating a construction site where workers need to build a building. The construction site has limited resources such as bricks, cement, and tools. Workers have specific tasks like laying bricks, mixing cement, and constructing the building. The bricks and the cement are transferred to the workers using a wheel cart which acts as a dispatcher. At a given time, only 1 unit of bricks or cement can be transferred to the workers, so the workers have to wait for sufficient units of bricks and/or cements before they start constructing the building. If the required units of cement/brick are unavailable, then the abundant resource cannot be produced until sufficient units of the second resource have been transferred to the workers. Your goal is to create an operating system-based simulation that efficiently manages these resources and coordinates tasks among the workers.

Tasks:

You have to do the following tasks for the simulation:

1. **Resource Management:**
    - Simulate the availability of construction resources such as bricks, cement, tools, and dynamically generated resources.
    - Implement resource allocation mechanisms to ensure controlled access by workers.
    - Introduce resource replenishment or degradation based on usage, external factors, or time-based decay.

2. **Process and Thread Management:**
    - Represent each construction task (laying bricks, mixing cement, scaffolding) as separate threads.
    - Simulate the creation and termination of threads based on the availability of tasks and resources.
    - Utilize thread synchronization mechanisms for coordinated access to shared resources.

3. **Synchronization:**
    - Develop synchronization mechanisms to manage access to shared resources like bricks, cement, and tools.
    - Ensure that simultaneous access to these resources does not lead to conflicts and implement synchronization to maintain data integrity.
    - Employ mutex, semaphores for synchronization.
4. **Memory Management:**
    - Implement memory management to store data related to the construction site, including resource availability, worker status, task progress, and now, dynamic adjustments based on task priorities.
    - Optimize memory usage for quick data retrieval and updates.
    - Utilize shared memory for inter-thread communication.

5. **Priority Scheduling:**
    - Implement a priority scheduling algorithm to efficiently handle critical tasks.
    - Define criteria for task prioritization, such as urgent repairs or specific construction phases.
    - Adapt dynamically to changes in task priorities based on real-time conditions.

Step 5.1 Assigning Priorities:

- Each construction task is associated with a priority level. Higher-priority tasks are assigned lower priority numbers.
- Priority levels are defined based on the criticality and impact of tasks on the overall project timeline.

Step 5.2 Example Priority Levels:

- High Priority Tasks: Urgent repairs, foundation laying, critical structural work.
- Medium Priority Tasks: General construction tasks, bricklaying, cement mixing.

- Low Priority Tasks: Non-critical tasks, finishing touches, aesthetic elements.

Step 5.3 Assignment of Priority Levels:

- Tasks are categorized into different priority levels during the project planning phase.
- Each task is tagged with its priority level, reflecting its importance to the construction project.

Step 5.4 Scheduling Logic:

- The priority scheduling algorithm selects the task with the highest priority for execution first.
- If two tasks have the same priority, other scheduling algorithms or criteria, such as first-come-first-served, may be employed as tiebreakers.

Step 5.5 Maintain Separate Task Queues:

- The system maintains separate task queues for each priority level (high, medium, low).
- Each queue contains tasks assigned to its respective priority level.

Step 5.6 Selecting Tasks:

- The system selects tasks from the highest priority queue for execution first.
- If the high-priority queue is empty, tasks from the medium priority queue are considered, and so on.
- The selected tasks are then assigned to available workers or resources for execution.

Step 5.7 Dynamic Priority Adjustments:

- The priority scheduling should be dynamic, allowing the system to adapt to changes in task priorities based on external factors or project requirements.
- If an urgent repair is identified or if there are changes in project timelines, the priority of tasks can be dynamically adjusted.

Step 5.8 Integration with I/O Simulation:

- Integrate the priority scheduling algorithm with an I/O simulation that mimics adverse weather conditions.
- During bad weather, certain tasks (e.g., outdoor construction) may need to be postponed, and the scheduling algorithm must adjust priorities accordingly.

Step 5.9 Consideration of Worker Skills:

- When assigning tasks, the system considers the skillsets of individual workers.
- Ensure that workers with the required skills are available for high-priority tasks to optimize efficiency.

Step 5.10 Efficient Resource Utilization:

- Optimize resource utilization by assigning high-priority tasks to the most skilled and available workers.

Step 5.11 Dynamic Changes in Priority:

- Allow for dynamic adjustments in task priorities based on real-time conditions.
- For instance, if an urgent repair is identified, it should be escalated to a higher priority, and the scheduling algorithm should adapt accordingly.

6. **I/O Management:**
    - Integrate I/O operations to simulate external factors affecting the construction site, such as weather conditions.
    - Simulate delays in construction due to adverse weather and coordinate worker activities accordingly.

7. **Error Handling:**
    - Implement robust error-handling mechanisms for unexpected scenarios, such as resource shortages, task conflicts, adverse weather conditions, and budget overruns.
    - Ensure that the system can recover gracefully from errors without compromising construction site integrity.

8. **Dynamic Resource Generation:**
    - Simulate the dynamic generation of resources over time.
    - Implement mechanisms for resource replenishment or degradation based on historical data and usage patterns.

9. **Dynamic Task Assignment:**
    - Develop algorithms for dynamically assigning tasks to workers based on their skills, proficiency, and the current construction site needs. - Consider factors like task dependencies and adjust task assignments accordingly.

10. **Realistic Worker Behavior:**
    - Simulate realistic worker behavior, including fatigue, breaks, and work shifts.
    - If a worker wants to get on break (due to any reason), then that worker is swapped out of the queue as it is useless for the processor now. For the replacement of the worker, you can use the LRU technique.

DELIVERABLES

- C/C++ code for both the questions.
- A detailed report outlining the design and implementation of the simulated construction site management system along with the custom shell system. The report must highlight the goals of both the questions, your approach to solve the problems, flowcharts/diagrams to illustrate the utilization of operating system concepts specifically dup() and dup2() system call, and work distribution between group members. The report must also highlight what have you learnt from this project and how this project can be improved with a brief conclusion of the content discussed in the report.

MARKING CRITERIA

Question Number	Tasks	Marks
Question 1	Standalone Commands	10
	Parametrized Commands	10
	Input Parsing and Tokenization	10
	Error Handling	10
	Process Creation and Control	10
	Synchronization	10
	Resource Management	10
	Single Pipelining	10
	Double Pipelining	15
	Command Parser for Pipelining	05
Question 2	Resource Management	20
	Process and Thread Management	35
	Synchronization	40
	Memory Management	35
	Priority Scheduling	30
	I/O Management	30
	Error Handling	20
	Dynamic Resource Generation	20
	Dynamic Task Assignment	20
	Realistic Worker Behavior	20
	Report (for both questions)	30
	Total	**400**

Good Luck!!!