CS 4323 Design and Implementation of Operating Systems I

Group Project: Full Marks 100
(Due Date and End Date: 04/21/2024, 11:59 PM CT)

This assignment is a group project that must be completed within the assigned group using either C or C++ programming language. If you are unsure about your group members, please consult the "Group Project" module located on the homepage of Canvas.

Students are encouraged to exercise their creativity and make assumptions, as long as they align with the requirements outlined in the assignment.

The project aims to provide students with practical experience in fundamental concepts of computer science and operating systems, including:

- Process management
- Interprocess communication using shared memory
- Synchronization techniques using monitors
- Deadlock prevention through resource ordering.

Students are required to view two videos relevant to this project. These videos can be found within the module labeled "Group Project." The first video, titled "Group Project Task Discussion" outlines the specific tasks to be undertaken within the project. The second video, titled "Group Project Challenges and Submission Discussions" addresses potential obstacles students may encounter during the project, provides guidance on project proceedings, clarifies expectations, and offers submission guidelines.

Project Problem:

The task for this project is to develop a program that effectively synchronizes access to shared files among multiple processes. To achieve this, your program must implement Monitors to prevent race conditions and ensure synchronized access. Additionally, to avoid deadlock situations, the program must enforce a strict ordering on resource allocation, thereby preventing circular wait scenarios among processes.

Project Description:

In a banking scenario, users can engage in various transactions, and among them, we focus on six key operations:

1. **Create an account:** This involves generating a new account in the system.
2. **Deposit:** Adding funds to an existing account.
3. **Withdraw:** Deducting funds from an existing account.
4. **Transfer balance:** Moving funds from one account to another.
5. **Balance inquiry:** Checking the balance of an account.
6. **Close an account:** Removing an account from the system.

In this setup, instead of utilizing a traditional database, students will interact with the file system. Each user's balance will be represented by a file in a directory. To ensure uniqueness, the account ID will serve as the file name. Each file will solely contain the final balance for the corresponding user.

The actions associated with the above listed transactions are as follows:

1. **Creating an account:** Involves generating a new file for the user.
2. **Deposit:** Adding the specified amount to the corresponding file.
3. **Withdraw:** Subtracting the specified amount from the appropriate file.
4. **Transferring balance:** Involves deducting funds from the sender's file and adding them to the receiver's file.
5. **Balance inquiry:** Entails reading the content of the file associated with the specified user.
6. **Closing an account:** Deleting the file corresponding to the user from the directory.

The input file contains information about user transactions and follows a specific format:

- The first row indicates the number of users attempting to perform transactions simultaneously.
- From the second row onward, transaction details are listed, with each row containing:
    - The account number (or user), as the first column.
    - The transaction type, as the second column, can be of following type:
        - **Withdraw:** Indicates a request to withdraw money from the account. In this case, the third column represents the withdrawal amount.
        - **Create:** Indicates a request to create a new account. The third column represents the initial deposit amount.
        - **Inquiry:** Indicates a request to check the account balance.
        - **Deposit:** Indicates a request to deposit a certain amount into the account. In this case, the third column represents the deposit amount.
        - **Transfer:** Indicates a request to transfer a specific balance to another account. The third column represents the transfer amount, and the fourth column represents the recipient account number.
        - **Close:** Indicates a request to close the account.

A sample input file is provided below for reference:

```
3
A332131 Withdraw 40
A342131 Create 40
A342131 Inquiry
A382131 Create 0
A382131 Deposit 10
A342131 Transfer 40 A382131
A342131 Inquiry
A382131 Inquiry
A342131 Close
```

This input file specifies the following information:

- **User 1 (A332131):**
  - Withdraws $40.

- **User 2 (A342131):**
  - Creates an account and deposits $40.
  - Inquires about the balance.
  - Transfers $40 to User 3 (A382131).
  - Inquires about the balance.
  - Closes the account.

- **User 3 (A382131):**
  - Creates an account with no initial deposit.
  - Deposits $10 into the account.
  - Inquires about the balance.

Note: If User 1 attempts a transaction on a non-existent account, the transaction will fail.

In scenarios where multiple users initiate transactions concurrently, it's essential to ensure proper synchronization. Concurrent transactions can lead to deadlock situations, characterized by circular wait conditions. To prevent deadlocks, it's critical to enforce a consistent and predefined order for resource lock acquisition. This ensures transactions proceed smoothly, minimizing the risk of circular waits and maintaining system integrity.

Project Components:

The project must incorporate the following components:

1. Process Management Module:
   - Utilize fork to create multiple child processes, each representing a unique user.
   - Allow each child process to execute one or more transactions.

2. Inter-process Communication (IPC) Module:
   - Create a shared memory segment accessible to all processes involved.
   - When a process executes a transaction, update the shared memory with transaction type and details, status, and timestamp.
     - For creating an account: update the shared memory with transaction type specified as “CREATE” followed by account number, initial deposit amount, "success" status, and creation date/time.
     - For depositing funds: update the shared memory with transaction type specified as “DEPOSIT” followed by account number, deposited amount, "success" status, and deposit date/time.

    - For withdrawing funds: update the shared memory with transaction type specified as “WITHDRAW” followed by account number, withdrawn amount, "success" status, and withdrawal date/time.
    - For balance inquiry: update the shared memory with transaction type specified as “INQUIRY” followed by account number, "success" status, and inquiry date/time.
    - For transferring funds: update the shared memory with transaction type specified as “TRANSFER” followed by transferred amount, sending and receiving account numbers, "success" status, and transfer date/time.
    - For closing an account: update the shared memory with transaction type specified as “CLOSE” followed by account number, "success" status, and closure date/time.
- After all child processes finish, the parent process should read the shared memory and display the results, containing transaction details as specified in the input file.

  Note: If any transaction fails, the shared memory should reflect a "failed" status instead of "success," with a reason specified. For instance, if a process attempts to withdraw $20 from a non-existent account, the transaction cannot proceed successfully. In such cases, the process should record that the transaction failed due to the non-existence of the account.

3. Synchronization Module:
    - Implement mechanisms for mutual exclusion and synchronization among processes accessing the shared file and updating the shared memory.
    - Utilize monitor-based synchronization techniques to prevent race conditions.
    - Display the queue status in the output when a new process enters the monitor queue.

4. Deadlock Handling:
    - Implement a deadlock prevention mechanism by eliminating circular waits.
    - Ensure program stability by preventing deadlock situations that may arise due to simultaneous access to shared resources by multiple processes.

Project Milestones:

1. **Week 1: Planning and Design**
    - Thoroughly understand the project requirements.
    - Design the project architecture, including data structures for the monitor queue, forks, synchronization strategy using monitors, and IPC mechanisms.
    - Allocate work distribution among group members.
    - Submit a progress report containing:
        - Project architecture design.
        - UML diagram illustrating work distribution among group members. (Sample UML diagram illustrated later)
    - Complete Self and Peer Evaluation Form.

2. **Week 2: Implementation**

- Implement the project architecture, including data structures for the monitor queue, forks, synchronization strategy using monitors, and IPC mechanisms.
- Develop file synchronization using fork for process creation and shared memory for data exchange.
- Create monitors to manage access to shared resources and ensure mutual exclusion.
- Implement deadlock prevention mechanisms.
- Submit a progress report containing source code.
    - Ensure the main() function is contained in a single driver file named driver.c or driver.cpp.
- Complete Self and Peer Evaluation Form.

3. **Week 3: Implementation**
    - Continue implementation from Week 2.
    - Submit a progress report containing updated source code.
        - Ensure the main() function remains in a single driver file named driver.c or driver.cpp.
    - Complete Self and Peer Evaluation Form.

4. **Week 4: Testing and Refinement**
    - Integrate individual work into the final deliverables.
    - Debug and refine the implementation to ensure correctness.
    - Submit the final report containing source code.
        - Ensure the main() function remains in a single driver file named driver.c or driver.cpp.
    - Complete Self and Peer Evaluation Form.

Deliverables:

At the end of each week (Week 1, 2, and 3), each group needs to submit a progress report containing the following:

1. Tasks Completed:
    - What work has been completed during that week.
    - For Week 2 and Week 3, provide the source code for completed tasks.
    - Include the screenshots of the test output demonstrating the completed tasks and explain.
2. Challenges:
    - Document any challenges encountered during the week.
3. Next Steps:
    - Outline the next steps planned for the project.

For the final deliverables (at the end of Week 4), each group should submit the following:

1. Source Code:
    - Provide the complete source code for the project.
2. Final Report:
    - Detail information about the design, implementation, and testing of the system.
    - Explain key algorithms and data structures used.

- Include test cases and screenshots of the results demonstrating the system's performance and reliability under various scenarios.

3. ReadMe.txt File:
    - Provide instructions on how to run the program on the CSX machine.
    - Specify the server on which the program has been tested.
4. Presentation Slides:
    - Prepare presentation slides summarizing:
        - Project objectives.
        - Contribution of each group member.
        - Methodology followed.
        - Test cases and results.
        - Conclusions drawn.

At the end of each week, each group member should submit the Self and Peer Evaluation Form as a separate document.

Grading Rubrics:

The grading will be as follows:

• Weekly Progress Report	[10 Points]
• Weekly Self and Peer Evaluation From	[10 Points]
• Successful process creation and implementing memory sharing between processes showing the transaction, status and modifications	[20 Points]
• Successful implementation of synchronization mechanism among processes using Monitor for file access	[20 Points]
• Successful implementation of deadlock prevention scheme	[15 Points]
• Implementation of queue in Monitor and display the processes in queue	[15 Points]
• Final Report	[10 Points]

Note:

- Each group member is responsible for ensuring the project and source code are not plagiarized.
    - Plagiarism will result in penalties as specified in the syllabus.
    - Plagiarized parts will be disregarded, and the project will be considered incomplete.
    - These guidelines are subject to the University's strict policy on plagiarism and will be strictly enforced.
- Failure of the program to compile or runtime errors on the CSX machine will result in a penalty of -20 points for each member.
- Any incomplete project will incur a penalty of -20 points for all group members.
- Issues arising from individual work not functioning correctly when merged into the group project will be considered as an incomplete project.
- Points distribution may not be equal among group members and will depend on individual responsibilities, task delivery, and contribution to the project

Grading Criteria:

This group project will be assessed based on students' ability to collaborate effectively as a team and successfully complete the assigned tasks. The emphasis will be on teamwork rather than individual efforts, with coordination among group members being crucial for achieving maximum points.

While there are weekly submissions, grades for these submissions will be finalized during the final project evaluation. This approach allows for an overall assessment of the project's progress and the contributions of each group member.

Submission Guidelines:

Weekly Submissions (For Week 1, 2 and 3):

1. Progress Report:
    - Each group must submit one progress report on Canvas under the link named "Progress Report (Week X)," where "X" represents the week number (1, 2, or 3).
    - The group should designate one member responsible for submitting the progress report on behalf of the entire group.
2. Self and Peer Evaluation Form:
    - Every individual group member must submit the Self and Peer Evaluation Form on Canvas under the link named "Self and Peer Evaluation Form (Week X)," where "X" represents the week number (1, 2, or 3).
    - It's crucial to emphasize that the Self and Peer Evaluation Form will be given significant consideration during grading since the TA and instructor may not have direct visibility into the project activities. Therefore, it's imperative for each group member to provide comprehensive information in their evaluations. Insufficient information may hinder the grading process and affect fairness. It is the responsibility of the group to ensure that all relevant details are provided in the final submission of the Self and Peer Evaluation Form.

Final Submissions (For Week 4):

1. Final Report:
    - The final report should be submitted by the group on Canvas under the designated link.
    - Ensure that the final report includes comprehensive information about the project activities and contributions of each group member.
2. Self and Peer Evaluation Form:
    - Every individual group member must submit the Self and Peer Evaluation Form on Canvas under the link named "Self and Peer Evaluation Form (Week 4)".

Source Code Requirements:

1. Individual files for each student's work:
    - Each student's work must be in a separate file.
    - Each file should include header information with the following details:
        - Group number
        - Author of the source file (or the code)
        - Email
        - Date
    - The code should include sufficient comments, with clear descriptions of each function, including input arguments and return values.
2. Code Functionality:
    - The code must function as a cohesive project when executed, as the TA will not run each individual student's work separately.

ReadMe.txt File:

- Include instructions on how to run the program on the CSX server. Provide all necessary details for running the program, assuming the TA has no prior knowledge.
- Specify on which CSX server the program has been tested.

Adhering to these submission guidelines ensures clarity, organization, and completeness in the project evaluation process.

Points to remember:

- Submit Reports and Evaluation Forms Promptly:
    - Avoid waiting until the last moment to submit reports or the evaluation form.
    - The due date and end date are the same.
    - If a group or member misses the deadline, the submission link will be unavailable.
- Submission Process:
    - Except for Self and Peer Evaluation Form (which needs to be submitted individually), all submissions need to be done as a group. So, only one group member needs to submit the progress report and the final report project on Canvas.
    - After submitting on Canvas, the submitting member must email the TA, cc'ing the instructor and all group members, to confirm the submission.
    - If any group member disputes the submission, they must inform theta, cc'ing the instructor, within 24 hours.
- Use of Group Page on Canvas:
    - A group page has been created on Canvas, accessible through "People" on the left side of the Canvas page.
    - It's highly recommended that all group members use this platform for communication and file sharing.
    - In case of disputes, information shared on this page will be used for resolution.

Illustration of UML:

Example Problem: Read the two integers from the files and perform various arithmetic operations like addition, subtraction, multiplication and division.

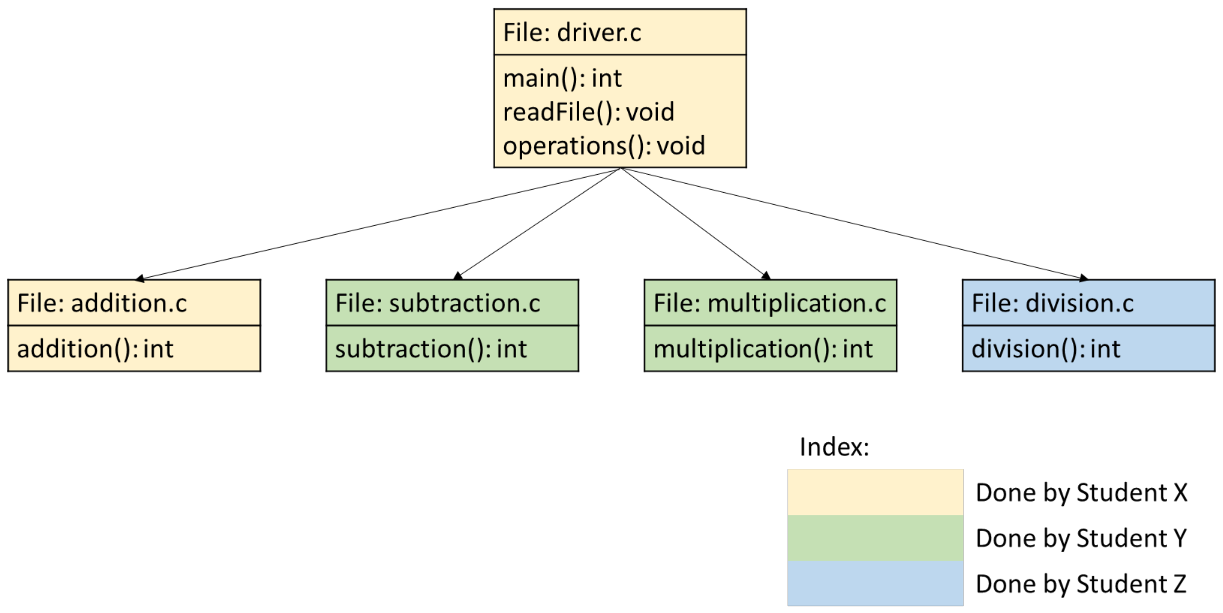

Fig. Illustration of UML for the given program

I believe that the UML is quite self-explanatory. But for the sake of completion, I will explain very briefly:

- 5 files: driver.c, addition.c subtraction.c multiplication.c and division.c
- Driver.c file has 3 functions:
  - main() has a return type of "int". It contains reading input file name from use which contains the data for various arithmetic operation. These data are passed to another function called readFile().
  - readFile() which has a return type void. It reads the data from the file and passes thesedata another function called operations().
  - operations() function has a return type void and calls all functions defined in anotherfiles: addition() from addition.c file, subtraction() from subtraction.c and so on.
- You got the idea for another files.
- There is a color coding used to explain who has done which part:
  - Student X has written code for driver.c and addition.c
  - Student Y has written code for subtraction.c and multipication.c
  - Student Z has written code for division.c