Progress Report: System Architecture and Module Development Week 2

Overview

We created the Input Parser, Process Manager, IPC Module, and the Synchronization Module. The driver.cpp can read a sample command file and print out which functions it recognizes.

Module Development Status

1. **InputParser Module**
    a. **Status**: Initial development phase completed. The module can read commands from text files and queue them for processing.
    b. Next Steps: add any additional commands if needed.
2. **Next Steps**: **ProcessManager Module**
    a. **Status**: Core process management functionality implemented, including starting, executing, and terminating processes.
    b. **Next Step:** Ensure integration with IPC and Synchronization.
3. **InterProcessComs Module**
    a. **Status**: Setting up functions in IPCModule.cpp
    b. **Next Steps**: Integrating Functions and class with the process manager
4. **SynchronizationMonitor Module**
    a. **Status**: Setting up functions in the SynchronizationModule.cpp
    b. **Next Steps**: Integrating Functions and class with the process manager
5. **DeadlockHandler Module**
    a. **Status**: Setting up functions in the Deadlock cpp
    b. **Next Steps**: Integrating Functions

How the driver.cpp works

Main Function UML Sequence Diagram Information

The sequence diagram for the main function will demonstrate the chronological interactions between the main function, the InputParser, and the ProcessManager, focusing on the process of loading, parsing, and handling commands.

Participants:

1. **Main Function**: The entry point of the application.
2. **InputParser**: Responsible for parsing commands from a file.
3. **ProcessManager**: Manages the processing of each command.

Flow:

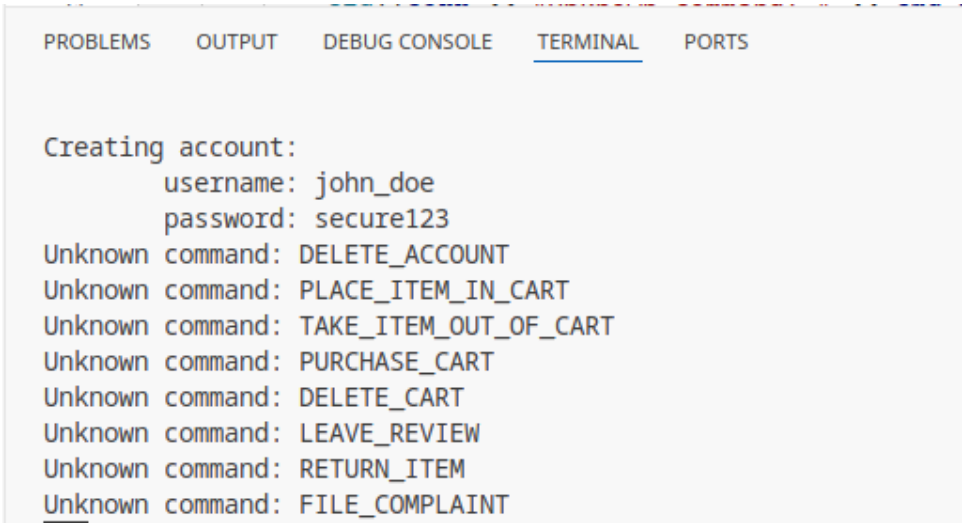
1. **Main Function** invokes InputParser::loadAndParse(filePath) to read and parse the command file.
2. **InputParser** processes the file and stores parsed commands internally.
3. Upon successful parsing, **Main Function** creates an instance of **ProcessManager**.
4. **Main Function** calls ProcessManager::processCommands passing the InputParser instance.
5. **ProcessManager** retrieves each command using InputParser::getNextCommand.
6. For each command, **ProcessManager** executes the corresponding command processing method based on the command type.

Simplified UML Sequence Diagram Notation

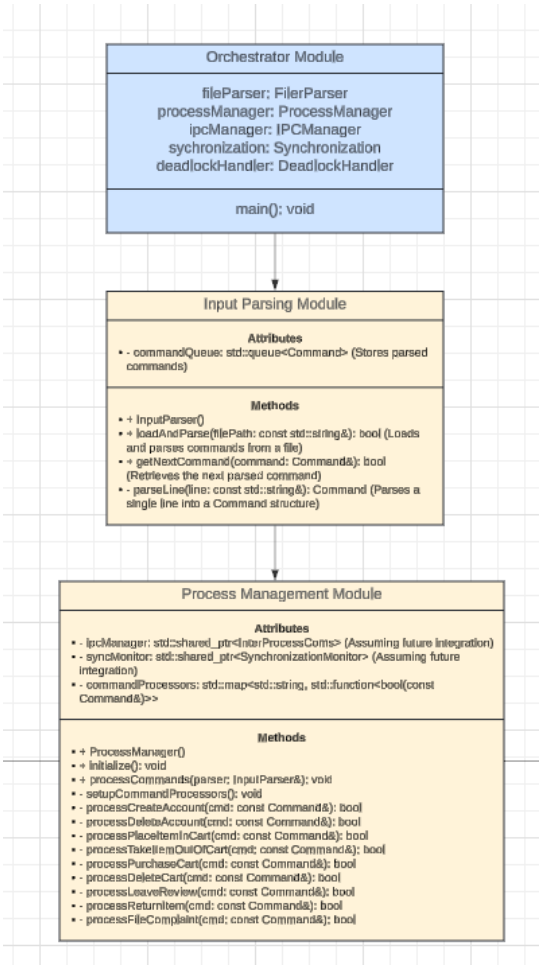
Considering the text-based format here, imagine the sequence as described:

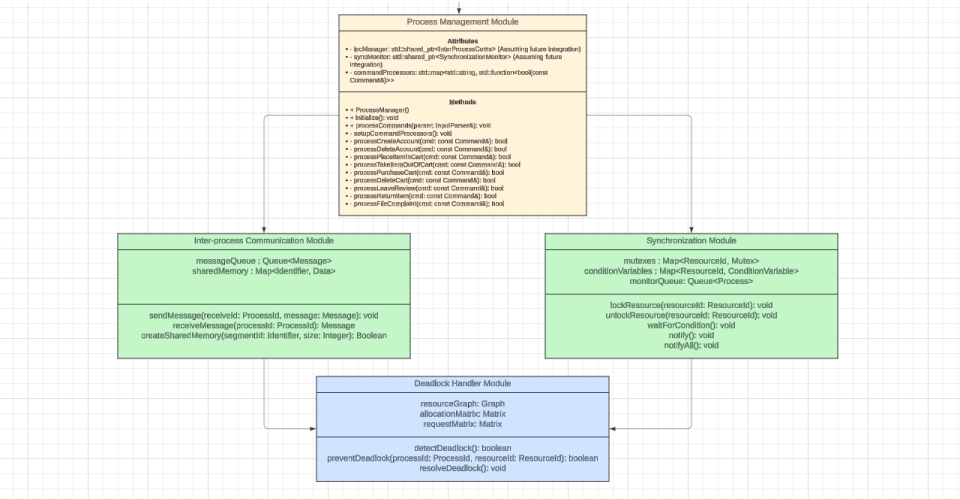
```
Main Function -> InputParser : loadAndParse("../text-files/commands.txt")
InputParser -->> Main Function : Parsing complete

Main Function -> ProcessManager : create instance
Main Function -> ProcessManager : processCommands(parser)

loop for each command
   ProcessManager -> InputParser : getNextCommand(cmd)
   InputParser -->> ProcessManager : cmd

   ProcessManager -> ProcessManager : process[CommandType](cmd)
end
```

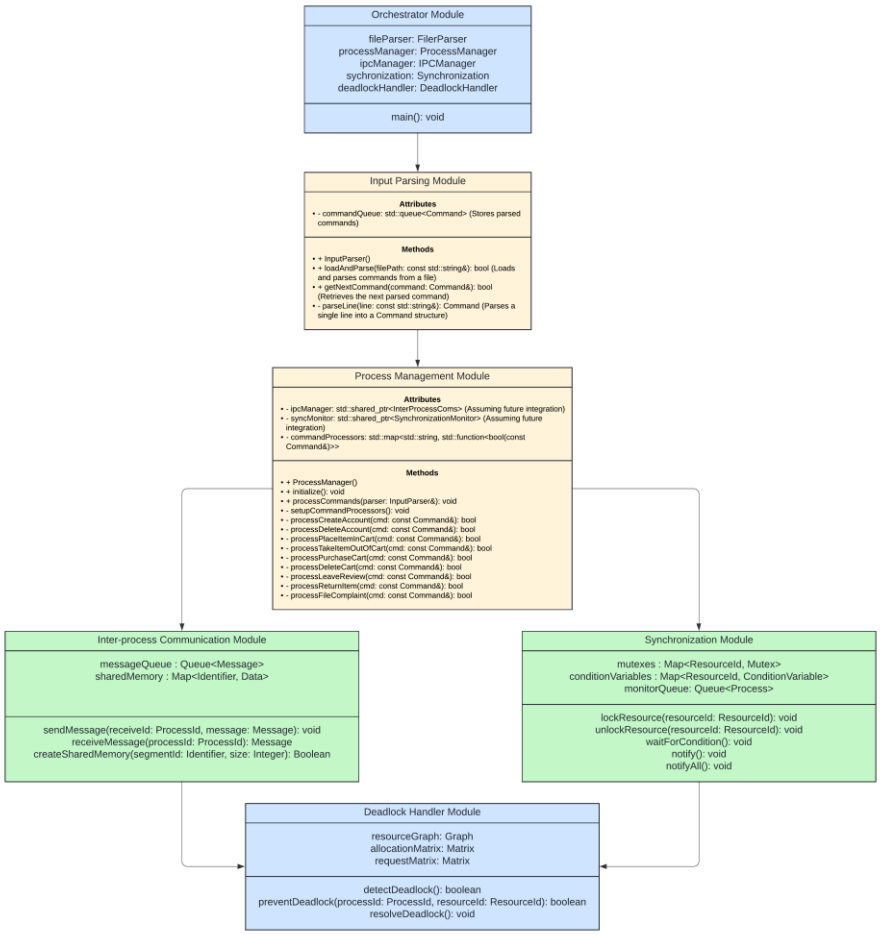
Running the driver.cpp program grants you this output



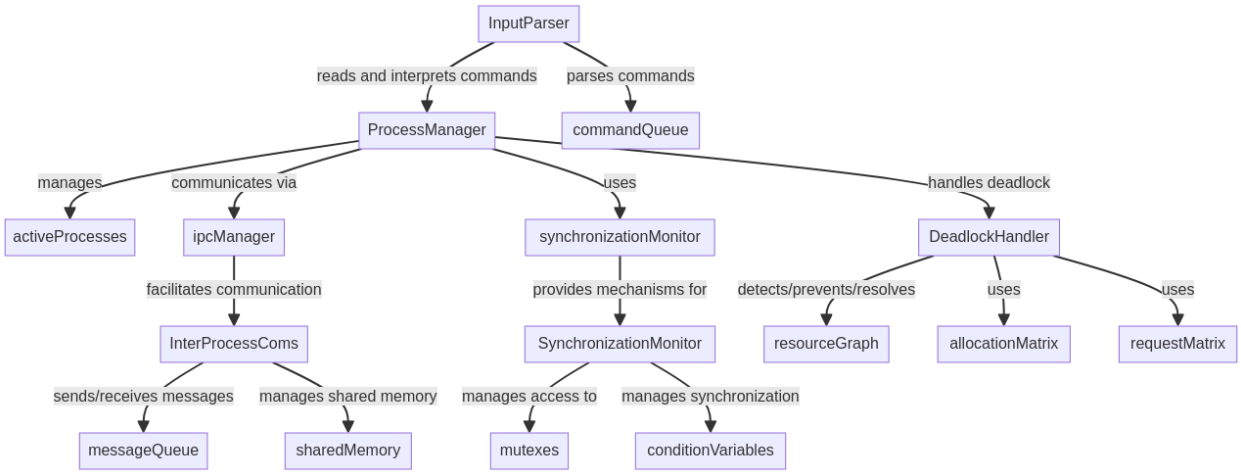
UML Diagram (Updated)







Architecture Diagram



Work Distribution

The team consists of three members: Thomas, Keelon, and Scott. Here's how the work has been distributed:

1. **Thomas**:
    a. **Responsibilities**: Lead the development of the InputParser and ProcessManager modules. Focus on enhancing the dynamic input handling capabilities and process lifecycle management.
    b. **Current Task**: Adding functions to the ProcessManager and the deadlock handeler.
2. **Keelon**:
    a. **Responsibilities**: Oversee the development of the InterProcessComs and SynchronizationMonitor modules. Key focus on robust inter-process communication and advanced synchronization mechanisms.
    b. **Current Task**: Creating the IPC and the Synchronization classes
3. **Ben**:
    a. **Responsibilities**: Develop the DeadlockHandler module with an emphasis on efficient deadlock detection and resolution strategies. Support integration with other modules for a cohesive system architecture.
    b. **Current Task**: Creating the deadlock handler class.

Challenges Encountered

Integration Complexity: Creating the commands for an ecommerce website. Trying to integrate the IPC and the Synchronization Module with the Process Manager.

Next Steps

Start the integration between modules.

Conclusion

Progress on the project is on track, with planning in module development being reached. The team's focus on the next phase will be on creating modules and their integration of each other and ensuring the system's robustness and reliability. Continuous collaboration and effective work distribution remain pivotal to our success.