Operating Systems

Assignment-2

Submission Date 31st March 2024

Instructions:

- *This is an individual Assignment.*
- ***All parties involved in any kind of plagiarism/cheating (even in a single line of code) will be given zero marks in all the assignments.***
- *Assignment deadline won't be extended*
- *Late submissions will be discarded so submit your assignment on-time*
- *You must follow below submission guidelines, otherwise your submission won't be accepted*
    - *create a new directory*
    - *change its name to the following format*
      YOURSECTION_ROLLNUMBER_NAME.
      E.g. A_22I-0012_Muhammad Ahamd
    - *put all your files into this newly created directory*
    - *Compress the directory into a compressed .zip file*
    - *Submit it on google classroom*
- *Use good programming practices (well commented and indented code; meaningful variable names, readable code etc.).*
- *Understanding the problem is also part of the assignment.*

QUESTION NO 1

Adventure Quest: Galactic Pursuit

Implement a simple multiplayer game where players can move around a game board and collect items using processes and pipes, a message-passing approach can be employed to ensure synchronization and coordination among players.

Game Board Generation:
Generate a random number between 10 - 99.
Multiply the generated number by the last digit of your roll number.
Divide your roll number by the generated number.
Take the mod of the result with 25. If the number is less than 10, add 15 to it. (it will make n)
Create an (n x n) board based on the calculated size.

Player Movement and Item Collection:
Create a separate process for each player, responsible for their movement and item collection.
Implement a message-passing system using pipes between the main process and player processes.

Message Passing:
Each player process sends messages to the main process when they move or collect an item.
Define a structured message format containing information about the player's actions (movement, item collection) and current position.
Use pipes to facilitate communication between the main process and player processes.

Main Process Logic:
The main process handles game logic, such as generating the game board, displaying the game interface, and updating scores.
Poll the pipes of each player process to receive updates.
Update the game state and scores based on the received messages.

Synchronization:
Since processes have their own address space, there is no need for synchronization techniques. However, proper synchronization is achieved through message passing.

Race Condition:
A race condition occurs when multiple processes try to access and modify the same game state simultaneously without proper synchronization. In this case, if two processes attempt to collect the same item simultaneously, it might lead to unpredictable behavior and inconsistencies in the game. For example, both processes might receive a message indicating that the item is available, causing both of them to collect it. This could result in duplication of items or other unintended consequences

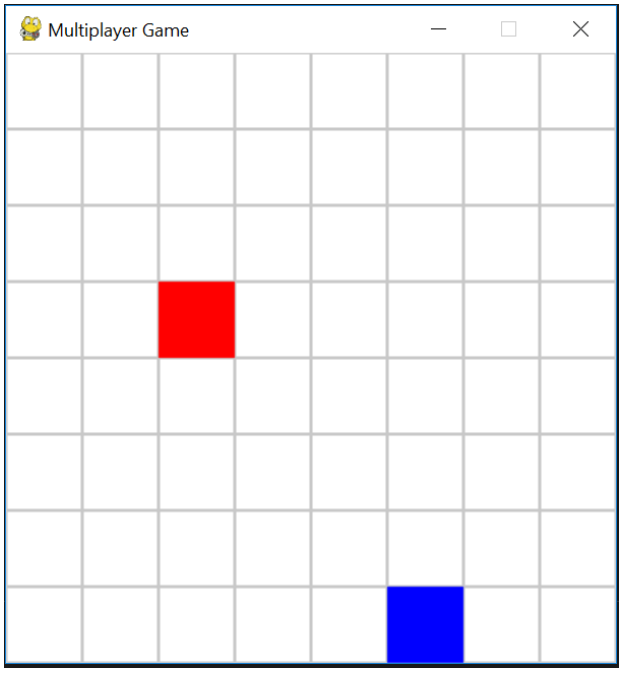
Message Ordering:
The order in which messages are received and processed by the main process determines the outcome. If one process sends a message to collect an item before the other process, it might successfully collect the item, leaving the second process to receive a message indicating that the item has already been collected. Alternatively, if the second process's message is processed first, it might successfully collect the item, causing the first process to receive a message indicating that the item is no longer available

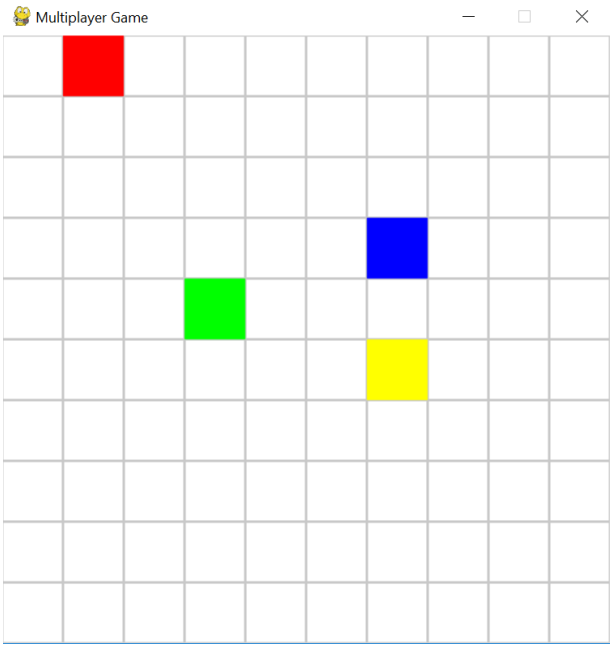
Priority Handling:
Consider introducing a scoring system for players.
Use the player's score or other game-related factors to prioritize handling messages from processes.
Processes with higher scores or better performance are given higher priority in the game.

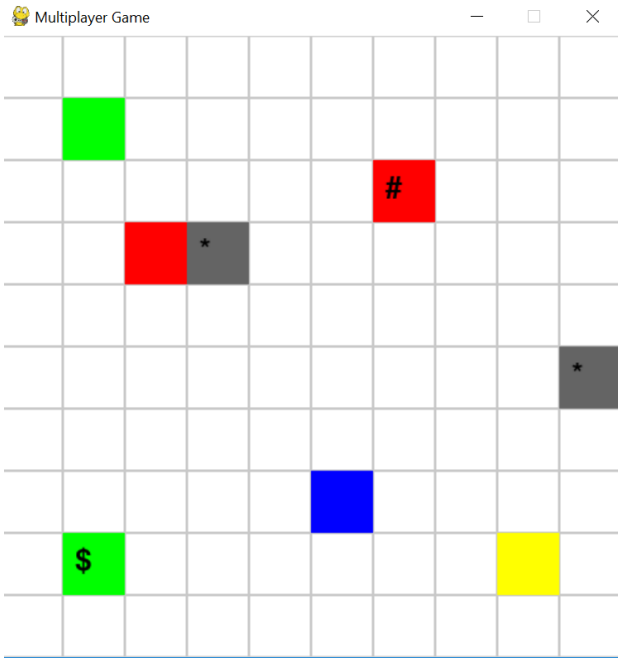
Final Note:
Utilize processes and pipes to achieve a parallel and message-driven architecture, ensuring that conflicts between player processes are avoided.
Leverage the independence of processes to enhance the overall concurrency and responsiveness of the multiplayer game.

Output

Your output screen can be either on terminal or using graphics (using graphics will lead to bonus marks). Each player can take only 1 step at a time.







QUESTION NO 2

ChatVista

Design a chat application that facilitates communication among multiple clients and a single server. Utilize fork and exec commands along with shared memory to implement the application. Ensure that the chat application meets the following requirements:

Client Features

- Clients can communicate with each other through the server.
- Clients can create group chats.
- Clients can choose the chat where they want to send messages.
- Clients can communicate privately with other users.
- Clients that create a group chat can also choose the other clients that they want to be added in the chat. And then make a request to the server.

Server Features:

- The server takes the number of clients as input and opens a new terminal for each client. // Use exec command for opening multiple terminals from one terminal.
- Each terminal displays the chat information of the respective client.
- The server manages group chat requests from clients.

- The server ensures that clients not belonging to a group cannot send or receive messages from that group.

Communication Mechanism:

- Shared memory is used for communication between clients.
- Clients send messages to the server, which forwards them to the intended recipient(s).

Implementation Guidelines:

- Use forked processes for managing client-server communication.
- You can define custom data structures for representing messages, group information, etc.

Additional Notes:

- Clients do not need server confirmation to receive messages.
- The server must handle client requests to create groups and manage group membership securely.
- Store all the messages of the chat in shared memory. When a new user sends a message it is appended to the previous chat which would be displayed to clients all the time.
- System should ask the user every time whether to read the chat or send the message.
- Consider implementing error handling and validation mechanisms to enhance application robustness.
- Ensure proper cleanup of resources after client disconnection or termination.

OUTPUT:

```
Server started with 3 clients.
Client 1 connected.
Client 2 connected.
Client 3 connected.
Client 1 requested to join Group 123
Client 1 joined Group 123
Client 2 requested to join Group 123
Client 2 joined Group 123
Client 3 requested to join Group 456
Client 3 joined Group 456
```

```
(Client 1 sends a private message to Client 2)
Client 1 to Client 2: Hi, how are you?

(Client 3 sends a private message to Client 1)
Client 3 to Client 1: Hey, let's discuss our project.

(Client 2 sends a group message)
Client 2 to Group 123: Meeting at 2 PM tomorrow.

(Client 1 receives private message from Client 3)
Client 1 received: Client 3 to Client 1: Hey, let's discuss our project.

(Client 2 receives group message)
Client 2 received: Client 2 to Group 123: Meeting at 2 PM tomorrow.

(Client 3 receives group message)
Client 3 received: Client 2 to Group 123: Meeting at 2 PM tomorrow.
```