# Operating Systems(CS)

Sunday, 2nd April, 2023

## Assignment# 02

---

- Zero marks will be awarded to the students involved in plagiarism.
- All the submissions will be done on Google classroom.
- You have to submit .cpp files in Zip Folder named after your roll no (20I-XXXX.zip). Naming convention has to be followed strictly.
- Be prepared for viva or anything else after the submission of assignment.

---

### Problem 1 –

You are tasked with implementing a simple multiplayer game where players can move around a game board and collect items. The game should have the following features:

1. The game board is represented as a two-dimensional grid of squares. Each square can either be empty or contain an item that the player can collect.
2. Players can move around the board by pressing arrow keys on their keyboard.
3. When a player moves onto a square that contains an item, they collect the item and gain points. There can be multiple players on the board at the same time.
4. The game should be able to handle multiple players moving and collecting items concurrently.

**You can calculate board size by following steps:**

1. Generate any random number between 10 - 99
2. Multiply the generated number by last digit of your roll number.
3. Now divide your roll number with the generated number.
4. Once division is done, Take the mod of respective number with 25. If your number is less than 10, add 15 to it.
5. You will receive a number less than 25, now create (n x n) board and start implementing your game.

To implement this game using threads, you can create a thread for each player, which handles the player's movement and item collection. Each player thread can be responsible for updating the player's position on the game board and checking if the player has collected any items.

To ensure that the game is thread-safe and that players do not interfere with each other's movement, you are not allowed to use mutexes or semaphores. However, one approach could be to use a message-passing system between the main thread and player threads. Each player thread could be responsible for sending messages to the main thread when they move or collect an item, and the main thread could update the game board and score based on these messages.

To avoid conflicts between player threads, each player thread could have its own message queue that it

sends messages to. The main thread could then poll these message queues to receive updates from each player, and update the game state accordingly.
This will ensure that only one player thread is updating the game board at any given time.
You can also use thread attributes to set the priority of each player thread, based on the player's score or other game-related factors. This will ensure that players with higher scores or better performance are given higher priority in the game.
Finally, you can use a main thread to handle the game logic, such as generating the game board, displaying the game interface, and updating the score for each player.

**---------- Best of Luck 😊 ----------**