Submission Date: 30th May 2022 till 11:55 PM

FINAL PROJECT

OPERATING SYSTEMS SPRING2022

INSTRUCTIONS

1. Plagiarism in course project will result in F grade in the course

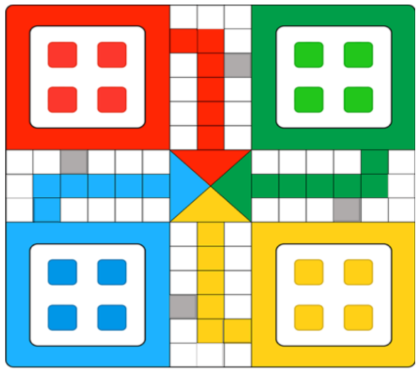
2. A maximum of two members in a group are allowed.
3. Make sure you submit your project at least 2 hours before the submission time. Late submissions won't be accepted even if they are late by just one minute.
4. Project must be submitted by only one group member
5. You can earn bonus marks by implementing extra features in the project.
6. Use good programming practices (well commented and indented code; meaningful variable names, readable code etc.).
7. Each file that you submit must contain names, and student IDs of both the group members on top of the file in comments.
8. Combine all your work in one folder and compress it into a zip file. The folder must contain .cpp files (no binaries, no exe files etc.).

9. Demos will be held after submission and projects will be graded individually
10. We may ask you to make changes in the code during demo so keep yourself prepared

Follow the given instructions to the letter, failing to do to so will result in a zero

Phase I (Ludo board game)

Ludo is a strategy board game for two to four players, in which the players race their four tokens from start to finish according to the rolls of a single dies. This project consists of designing a multithreaded application of Ludo for four players each having maximum four tokens. Two, three or four can play, without partnerships. At the beginning of the game, each player’s tokens are out of play and staged in the player's yard (one of the large corner areas of the board in the player's color). When able to, the players will enter their tokens one per time on their respective starting squares, and proceed to race them clockwise around the board along the game track (the path of squares not part of any player's home column). When reaching the square below his home column, a player continues by moving tokens up the column to the finishing square.



The rolls of a single die control the swiftness of the tokens, and entry to the finishing square requires a precise roll from the player. The first to bring all their tokens to the finish wins the game. The others often continue play to determine second, third, and fourth place finishers.

Rules:

• Each player rolls the die.
• Players do not take turns sequentially i.e. players alternate turns in a random fashion. But for each round of rolling the die to be complete, each player needs to complete his turn.
• To enter a token into play from its yard to its starting square, a player must roll a 6. If
the player has no tokens yet in play and rolls other than a 6, the turn passes to the
next player. Once a player has one or more tokens in play, he selects a token and moves it

forwards along the track the number of squares indicated by the die. Players must always move a token according to the die value rolled.

• Passes are not allowed; if no move is possible, the turn moves to the next player.

• A player will keep on throwing the dice if Six comes.

• Three consecutive sixes will result in loss of turn and all his numbers in that turn will be discarded.

• When a 6 is rolled, the player may choose to advance a token already in play, or may enter another staged token to its starting square. Rolling a 6 earns the player an additional or "bonus" roll in that turn. If the bonus rolls results in a 6 again, the player earns an additional bonus roll. If the third roll is also a 6, the player may not move and the turn immediately passes to the next player.

• Players may not end their move on a square they already occupy. If the advance of a token ends on a square occupied by an opponent's token, the opponent token is returned to its owner's yard. The returned token can be reentered into play only when the owner rolls a 6.

• There are some "safe" squares on the game track which protect a player's tokens from being returned. They are shown in the board with player colors. A player's home column squares are always safe, however, since no opponent may enter them.

• If a piece lands upon a piece of the same color, this forms a block. This block cannot be passed or landed on by any opposing piece.

• When a piece has circumnavigated the board, it proceeds up the home column. A piece can only be moved onto the home triangle by an exact throw.

• A player should not be allowed to enter into his Home column until he has removed at least one opposing piece.

• The first person to move all 4 pieces into the home triangle wins

There are several different ways of multithreading this application. One suggested strategy is to create threads that check the following criteria:

• Four threads are created and will be assigned to each player.

• A thread to check each row and column of the grid to find out the token and player to be passed for completion. This thread should run randomly to check the hitting token and victim player.

• One thread known as “Master thread” should track for all four players of hit record as no token can enter into its home column if its hit rate is 0. The property of this thread is it can cancel all the other four threads in certain situations. For example: The stopping criteria or thread cancelation will be as follows:

o For consecutive 20 turns if any player can’t get a 6 in a die or cannot hit any opponent player, he/she will be kicked out from the game. (Cancelled by this thread).

o When all tokens of a player reach its home, it will send the signal to this

master thread about its completion. The master thread will verify and then
kill the respective thread after calculating its position.

Passing Parameters to Each Thread

The parent thread will create the worker threads, passing each worker the location that it must check in the grid. This step will require passing several parameters to each thread. The easiest approach is to create a data structure using a struct. For example, a structure to pass the row and
column where a thread must begin validating would appear as follows:

```
/* structure for passing data to threads */
typedef struct
{int row;
int column;
int hit_record //optional
}
parameters;
```

Pthreads program will create worker threads using a strategy similar to that
shown below:

```
parameters *data = (parameters *) malloc(sizeof(parameters));
data->row = 1;
data->column = 1;
/* Now create the thread passing it data as a parameter */
```

The data pointer will be passed to either the pthread create() (Pthreads) function, which in turn will pass it as a parameter to the function that is to run as a separate thread.
Returning Results to the Parent Thread
Master thread is assigned the task of determining the winners/victim of a particular region of the Ludo grid. Once a player hits opponent token it must pass its results back to the parent.
In case of thread cancelation, it should be informed to the parent thread

Phase 1:

You are required to create the grid in this phase. Grid and dice variable should be global and considered as two shareable resources. But it is mandatory that each thread can access single resource at a time. E.g., If first player is rolling the dice, it should release the dice resource before accessing the Ludo board/Grid. Complete structure for the game will be part of this phase.

EXPECTED OUTPUT: Phase I

Main thread will display:

• Complete grid for Ludo board and token placed in their home yards.

Phase 2

In this phase, you have to give complete Ludo game implementation.

Conditions:

Considering the below conditions:
Token must be implemented using semaphores. They are varying when a resources relies and has been added after dice rolling.

• You have to take user input for number of tokens (maximum 4 and at least 1). This input will be take only once and assigned to each player.
• Turn will be followed randomly for four players. At each iteration, all players will try to access the dice. The one who get the dice will roll it and calculate its turn value.
• Both dice value and turn will be calculated randomly but in one iteration every player should get its turn.
• Player have their own safe squares, where no opponent can hit them. (Clearly, shown with colors in figure 1).
• When there is hit rate 0 of any player and it is near their home side, they are not allowed to enter in home, but complete another circle.
• After rolling of dice each player will release dice resource and then are able to get Ludo board resource. The tokens on the board are moved according to the random numbers of dice from (1 to 6).

Note: Here you have to take care of mutual exclusion problem. E.g., if player one rolls the dice and get Ludo board access, at that time he has to move 4 steps onward. At the time, another player rolls the dice and get the board resource first. (as player 1 should get it first). Considering they both are having the same final place in the grid after their respective turn, there will be hit condition occurred. Player 1 will be hit by player 2 in correct sequence, but if player 2 get first access it will be hit by player 1 (which is wrong).
Hint: You can use conditional variable previous turn here. So, that the player who has the last turn will get access of the Ludo board first.

EXPECTED OUTPUT: Phase 2

Main thread will display the following results:

• Grid should be display after each iteration. Each token should be placed for every player.
• You have to display one or two grids at a time on each window.

Hint: You can use sleep here.

• Winner of the game as 1st, 2nd, 3rd.
• Cancelled threads id and player names.
• Number of tokens.
• Hit rate of each player

Bonus Feature:

1. Add a GUI for Ludo grid that will simulate the real game.
2. Players should be able make teams and play with each other. Rest of the rules should be the same except the ones given below:

a) If the pieces of different members of a team land on each other than that will create a

block. . This block cannot be passed or landed on by any opposing piece.
b) A team will only win when all the 8 pieces of both members move into home triangles.
c) If one member of a team successfully moves all his 4 pieces into the home triangle then he should be able to share his roll and move the pieces of his partner. He should start sharing after throwing a Six.
You may think of other interesting (and programmatically challenging) features to implement to get bonus marks. The decision to give the bonus marks for that feature will be with the teacher so you should discuss any idea with your respective teachers to know whether it qualifies as bonus or not.
To be eligible for Bonus marks, you will need to first complete the phase 1 and phase 2.

Project Report Requirements

• Provide Pseudo codes for Phase I and II.

• Appropriate illustrations of the operating system concepts you used in your Pseudo codes.

• Provide your implemented codes.

• Provide your system specifications.

• Short paragraph how could you implement these concepts in some other scenario. (at least one)

Other Instructions:

Use any appropriate and efficient synchronization technique for this project. The code must be properly commented and this carries marks too. Group details should appear on separate page in report. You have to mention the contribution of each group member for the project. Again, all this carries marks.

Happy CODING ☺