THREAD POOLS AND ASYNCHRONOUS PROGRAMMING

1 OBJECTIVES

The main goals of this assignment are to learn and implement the follow concepts of multithreading:

- Thread Pools
- Tasks as worker threads
- Asynchronous programming which a part of thread pool implementation

The above topics are examined using the modern Java/C# techniques.

2 DESCRIPTION

This assignment consists of three alternatives. You should implement at least one of them:

2.1 **Client-server chat program**: an application allowing a client to chat to the server. Some help and guidance on how to establish a network communication in C# is provided at the end of this document.

2.2 **A simple pool game**: simulate a primitive pool game – a challenge!

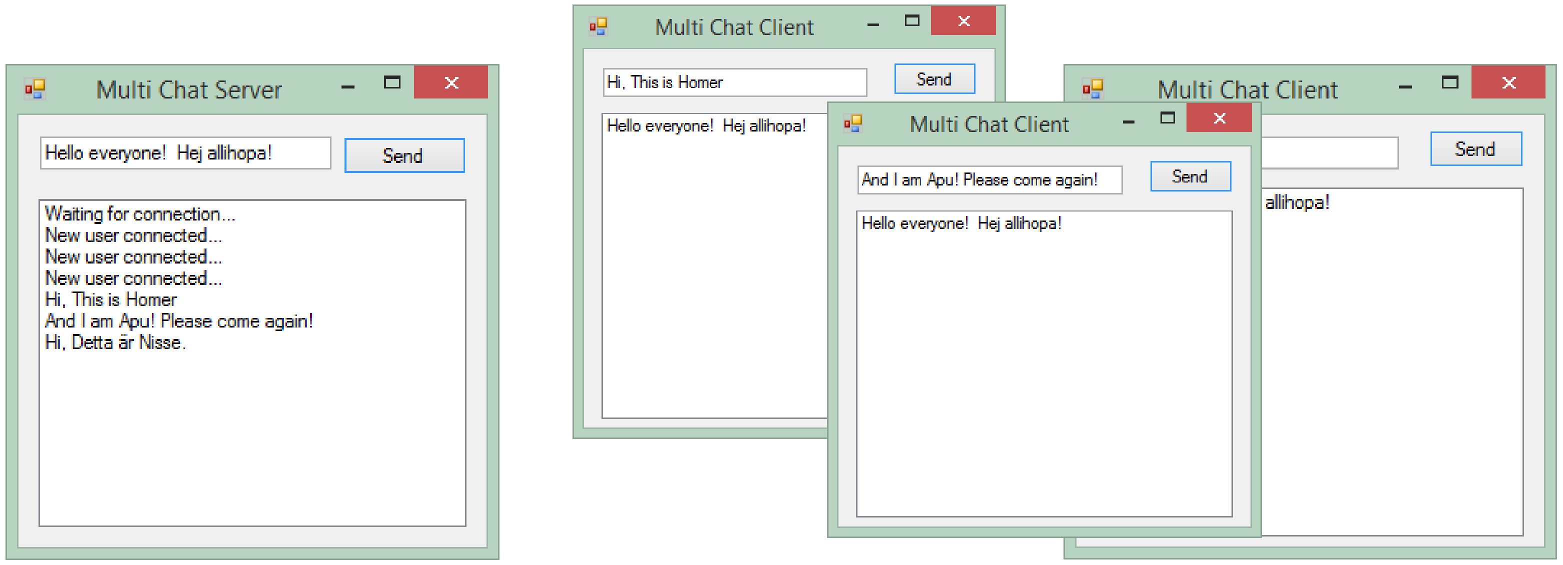
2.3 **Car race**: simulate a racing match with cars running (competing with each other) at the same time.

For the alternative you choose, you are to use thread pools and tasks in your programming.

3 Client-Server Chat Program

The chat application consists of two applications, a server part and a client part. These become two separate projects, one for the server application and the other for the client application. Here is an outline of the features expected:

3.1 It should be possible for more than one client to connect to the server and chat to the server at the same time. You can think of the server application as someone who runs an interactive blog. The server application should display every text it receives from its clients.



Multi Chat Server input: Hello everyone! Hej allihopa!

```
Waiting for connection...
New user connected...
New user connected...
New user connected...
Hi, This is Homer
And I am Apu! Please come again!
Hi, Detta är Nisse.
```

Multi Chat Client input: Hi, This is Homer

```
Hello everyone!  Hej allihopa!
```

Multi Chat Client input: And I am Apu! Please come again!

```
Hello everyone!  Hej allihopa!
```

Multi Chat Client:

```
allihopa!
```

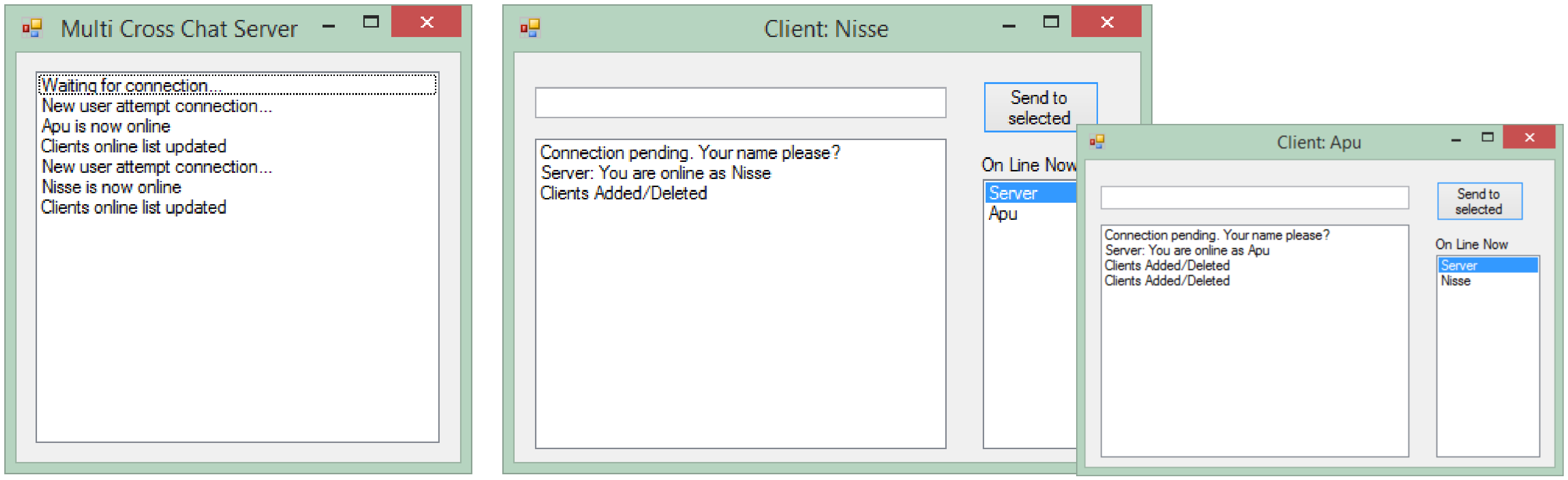
3.2 To simplify the work, the clients need not to communicate with each other (through the server of course). It is sufficient that each client communicates with the server, the server displays the messages it receives from all clients, and it sends a reply message to each client. The reply message is prepared by the user using for instance a text box as in the above figure. When the two applications (Server and Client) are completed, they can be tested as follows.

3.2.1 Run the server first(!), by either starting the exe-file (under the bin/Debug) or start from the Visual Studio (VS)

3.2.2 Start multiple versions of the client application using the exe-file (or by starting two or more instances of VS).

3.3 The server should assign a separate thread from the thread pool to run each client instance.

3.4 Optional: If you would like to develop a chat program in which one client can send and receive a message from another client (without others seeing the communications), you may apply your own design. You can for example let every client see which other clients are online, select one and then send a message. Each client should also be able to communicate with the server as in above.



```
Waiting for connection...
New user attempt connection...
Apu is now online
Clients online list updated
New user attempt connection...
Nisse is now online
Clients online list updated
```

```
Connection pending. Your name please?
Server: You are online as Nisse
Clients Added/Deleted
```

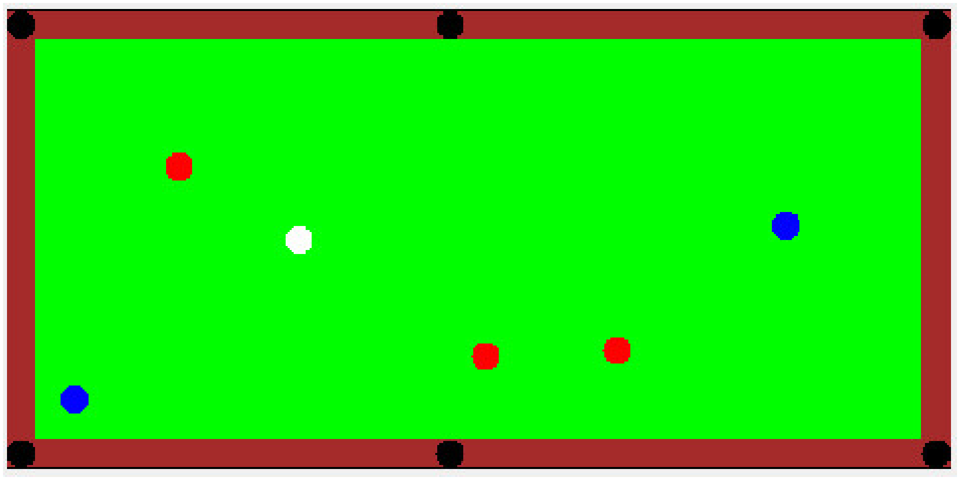
Server
Apu

```
Connection pending. Your name please?
Server: You are online as Apu
Clients Added/Deleted
Clients Added/Deleted
```

Server
Nisse

4 A Simple Pool Game

The Pool game is built using graphical panels. As this can be done in many ways, no GUI is given and is left for you as a small challenge. In the sample GUI below, the outer panel defines the boundaries of the table and the inner panel sets the boundaries for ball movements, i.e. the play area. Between the panels, six holes serving as targets are drawn, as shown in the image below. You can follow the steps below to get started:



4.1 Draw the table as described above or according to your own design.

4.2 Draw and place about six balls randomly on the game panel, the inner panel.

4.3 The plays starts by the user selecting a ball. The selection can be programmed as input from the keyboard or by a mouse-click, whatever is easier for you. The center of the ball determines the start point.

4.4 When a ball is selected, change its color to white or gray to mark the selection.

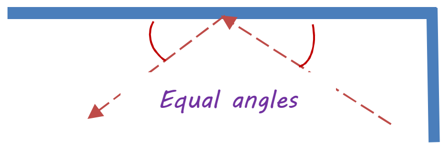
4.5 The user then marks another location on the table (inner panel). The line between the start point and this point makes the direction of the ball to roll on.

4.6 The ball can then start moving by another interaction, for instance when the user presses the Enter key, or double-clicks on the mouse. Assume that this interaction simulates the action of a ball being hit by a billiard cue.

4.7 The ball that is hit either collides with another one or hits the edges of the table. The program should detect these events.

4.8 For simplicity, assume that every move is limited to a distance about twice the width of the panel. In addition, you can assume that every hit on another ball changes the direction of the moving ball with +45 degrees, and the ball that is being hit starts rolling in the same direction as the moving ball, with an angle of -45 degrees. The direction of a ball hitting a wall can be mirrored, i.e. the ball returns and moves in a direction with a symmetrical angle as demonstrated in the image below.

4.9 When a ball hits a target (a hole), the thread stops and the ball is removed from the playing area.



4.10 Use a thread pool for the tasks that happen on the billiard table. The ball that is hit by the cue is to be executed by a thread from the thread pool. When the starting ball comes into a collision with another one, the second ball is taken care of another thread from the thread pool and if the second ball collides with a third one, a new thread is to take the ball and so on.

4.11 Every move should be a task for a thread from the thread pool.

4.12 You can decide the rest of the application and take you own decision on the design and the solution by yourself. However, it is sufficient if you manage to program the above steps.

5 The Car Race

This alternative is to simulate a racing track, by adding cars to the race. Each new car starts in a task in a thread pool. The car MUST have an ID, a fuel consumption and a tank volume. The fuel consumption and tank volume is chosen from a list, with a default value selected. The ID must be provided before the car can be added to the race. Every car then opens a form/window where the user can start/pause/exit and change the speed. Every car has a status and at regular intervals, the cars report their ID, driven distance and status. No consideration to acceleration, the speed is read at start, and after a report, assuming constant in between. The distance an fuel remaining is calculated, and if the car runs out of gas, reporting is stopped and a message box shows the "Car xx is now refilled" and after user OK the car is restarted, reporting again. The following details simulate the problem:

5.1 Consider a maximum of 10 different fuel consumptions in the range 0.4 to 1.2 l/mil. And some tank volumes from 20 to 50 l.

5.2 Make the reports every 5 seconds, representing 0.5 hours driving.

5.3 Start reporting after start, restart (after filling fuel) and stop on paus and out of gas. The exit button removes the car from the race and stops the task in the thread pool.

5.4 Use an enum for Car status: Not started, Running, Paused, Out of gas, Leaving and restarted. Once the car is entered is is not yet started, but log this as car ID, 0 driven km and status Not Started.

5.5 The speed control can be from 10 to 100 km/h.

5.6 The controls can be others than shown below, just visualize the idea.

5.7 The Car forms shall only be closed by the Exit Race button, not by the default button.

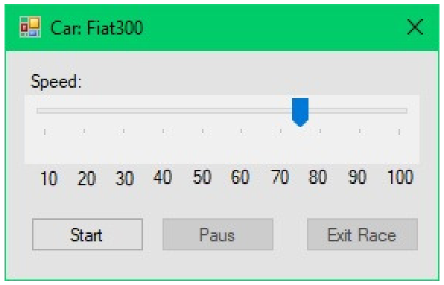
6 Specifications and Requirements for a pass grade (G)

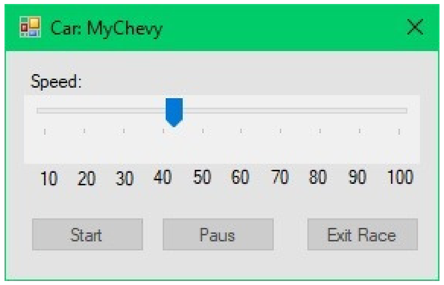
6.1 To qualify for a pass-grade, you should implement at least one of the above alternatives with good code quality.

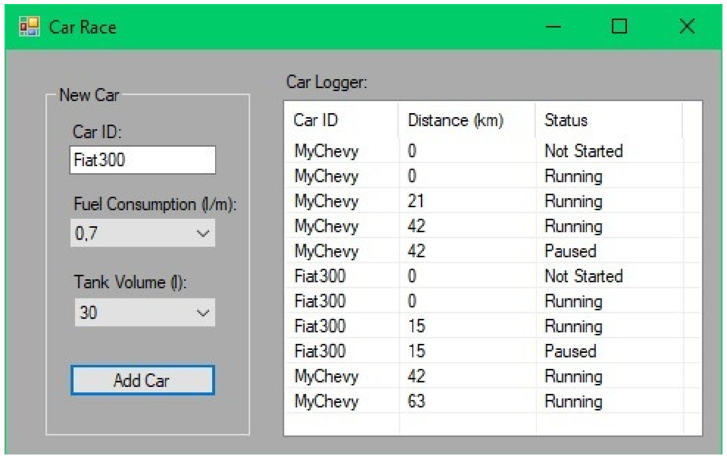
6.2 Do your programming work well-structured, well organized and always have OOP in mind. Use proper variable and method names, document your code by writing comment in your code.

6.3 You may certainly bring changes in the application to make it more it more fun and full-featured.

6.4 Test your application carefully before submitting.







Car ID	Distance (km)	Status
MyChevy	0	Not Started
MyChevy	0	Running
MyChevy	21	Running
MyChevy	42	Running
MyChevy	42	Paused
Fiat300	0	Not Started
Fiat300	0	Running
Fiat300	15	Running
Fiat300	15	Paused
MyChevy	42	Running
MyChevy	63	Running

7 Grading and submission

Submit your solution to Canvas as before. After or before submitting your assignment to the module, show your assignment to your lab leader during the scheduled hours in the labs.

Good Luck!

Farid Naisan,
Course Responsible and Instructor