CITS 3002
Computer Networks

Assignment: Socket Programming

Due date: Tuesday 8pm, 21 May 2019
Drop dead due date: Friday 8pm, 24 May 2019 (10% penalty on raw grade per day)
Worth: 20% of the whole grade
This assignment is to expand the knowledge on socket programming. You are required to research further on socket programming in addition to the materials provided in this unit in order to complete the assignment. Please set aside enough time to finish this assignment rather than leaving it too late, as it will require some time to complete.

1. Intro: Electronic Farce presents: RNG Battle Royale

Congratulations! You've been given an internship at one of the larger game developers, *Electronic Farce*, who have just launched yet another battle royale. The company intended all the interns to be concerned with maintenance, testing and bug-fixing, but due to budget and time constraints a larger problem has arisen.

They forgot to implement a server

In a rather spectacular but not altogether surprising fashion, EF have tasked all of the interns (you) to implement their server in a short period of time. Your task is to write a basic server in C, which adjudicates and runs an RNG battle royale game. Of course, you will need to write some players (for example, in Python) to test your server as you go. We provide more specific game details below.

2. The Game

The game in question is a rather simple dice game. A game consists of multiple rounds, all players start with some (non-negative) number of lives. In each round, each player makes a guess on the outcome of the next dice roll. The server 'rolls' two dice. Players who correctly guess the outcome advance to the next round without losing a life, while the rest lose a life. Players whose lives are reduced to zero are eliminated from the game. The last player remaining is deemed the victor. Figure 1 shows the game flow.

The game play

Players are given X lives (decided by the server. E.g. 3). Upon losing their lives they are knocked out of the game. Given two six sided dice, players are allowed to make the following moves:

- Even - The sum of the two dice is an even number (excluding doubles)
- Odd - The sum of the two dice is an odd number greater than 5
- Doubles - The two dice rolled are the same
- Contains N - At least one of the dice rolled in N

If the guess was incorrect, the player loses a life. If the player's lives are reduced to zero, the player is out of the game.

CITS 3002
Computer Networks

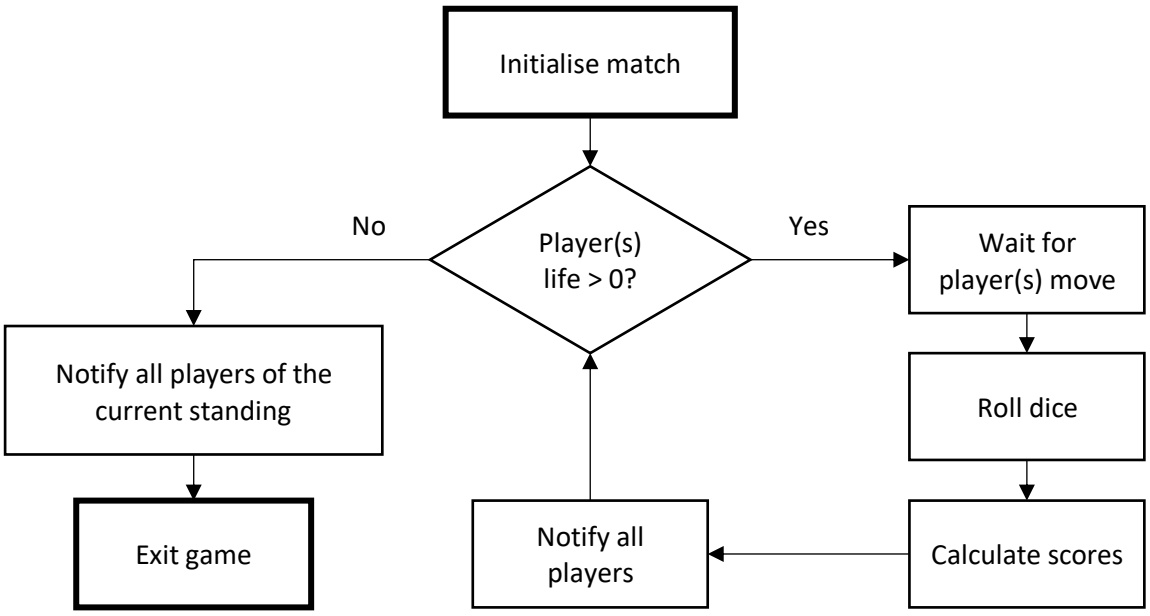

Figure 1. Game flow (single game)

Network Requirements

For sake of simplicity we specify the format of packets to be sent below.

Client to Server (the first field, %d, refers to client_id provided by the server).

- "INIT"
- "%d,MOV,EVEN"
- "%d,MOV,ODD"
- "%d,MOV,DOUB"
- "%d,MOV,CON,%d" int[1-6]

Server to Client

- "WELCOME,%d" - Sent to acknowledge a connection, provide client's 3 digit ID
- "START,%d,%d" num_players, lives - Sent to notify that the game has begun
- "%d,PASS" - Sent to players who were correct
- "%d,FAIL" - Sent to players who were wrong
- "%d,ELIM" - Sent when a player's lives are reduced to 0
- "VICT" - Sent when no players are left
- "REJECT" - Sent to players trying to connect too late
- "CANCEL" - Sent when a game cannot start

CITS 3002
Computer Networks

Game summary (repeated with networking)

We can now describe the game-flow with respect to the packets sent between client and server.

```
Client->Server "INIT"
  if(allowed)
    Server->Client "WELCOME"
  else
    Server->Client "REJECT", exit()
// Some time will elapse
if(enough_players)
  Server->All_Connected_Clients "START,%d,%d"
else
  Server->All_Connect_Clients "CANCEL", exit()
while(true){
  Client->Server "MOV, EVEN||ODD||DOUB||CON,%d"
  if(Server->Client "PASS" || "FAIL")
    continue;
  elseif(Server->Client "ELIM"||"VICT")
    exit()
}
```

Packet Definition

For simplicity and consistency, we define the packet you should use below. It consists of:

- A character array of length `14`

The structure comes from the limited cases allowed in the string.

The client to server packets are of the form:

- "INIT" - Used to connect to a game
- "%d1,MOV,EVEN||ODD||DOUB||CON,%d2" (%d1 client_id, %d2 being a 1 digit integer) - Representing a move in the game

The server to client packets are of the form

- "WELCOME,%d" - Sent to a client if their connection is accepted, %d being a 3 digit client id.
- "REJECT" - Sent to a client if their connection rejected
- "START,%d,%d" (%d being a 2 digit integer) - Sent when a game has begun, the first number is the number of players in the match. The second is the number of 'lives' you start with (can vary per game)
- "CANCEL" - Sent when not enough players join a match to begin
- "%d,PASS" - Sent to players whose moves are correct, %d client id
- "%d,FAIL" - Sent to players whose moves are incorrect, or failed to make a move. These players are still in the game however, %d client_id
- "%d,ELIM" - Sent to players whose lives are depleted, %d client_id
- "%d,VICT" - Sent to the last remaining client (instead of a 'PASS' or 'FAIL' packet), %d client_id

Edge-cases

In the case of a draw (i.e. 2 players are removed simultaneously), then both players win.

3. Tasks

Your goal is to implement a server in C which implements the game described above, then shuts-down.

- Clients connect to this server and are updated on game state and asked to make moves
- Emphasis is on the ability to start, maintain and tear-down connections in C versus optimising network performance

To get started, you are provided with:

- `basic_server.c` - A basic server written in C. This is a starting point only, you are allowed to modify it as you see fit (we provide some starting suggestions).
- `socket_client.py` - A basic client written on Python. This is a starting point only to show how we expect connections to be setup. You are expected to write your own clients while testing.

To guide your efforts, we provide three 'tiers' of tasks to complete. Solutions which meet Tier 1 requirements are minimally viable, higher tiers offer more granulated mark allocations for solving more complex problems. We will only assess a single submitted implementation (i.e. the tiers 'stack' on each other rather than implementing the requirements of each separately).

Tier 1 - If everything works as you expect

Implement a basic server in C which plays a single game of our game with a single player and then tears itself down. All players are expected to behave themselves (only make legal moves) and connections are expected to be fast and stable.

- Server can receive a client connection and send a game initialisation message
- Update the client that the game has started
- Receive moves from players (allowed a basic timeout for moves). If a player fails to make a move, they lose a life
- Updates game-state based on moves from players
- Updates players on the outcome of a round
- Is able to tear down a finished game

Assumptions: Players know the IP address and expected port number of the server.

Tier 2 - Scaling up

In addition to all Tier 1 requirements:

- The server must be able to play a game with at least four players in a game
- The server must inform each player only about their own game state.

CITS 3002
Computer Networks

Tier 3 - Connection issues

Of course, the world is not perfect. In addition to all Tier 1 requirements, the server must be able to handle unexpected network behaviour.

- The server handles players exiting mid-game
- The server must handle players attempting to join mid-game
- Server must not send game-state information to this player

Assumptions: If the winning player disconnects, behaviour is undefined.

Tier 4 - Game logic extended

Even when network failures are considered, players can still behave badly.

- The server must handle incorrectly formed move packets gracefully
- This includes 'valid' but impossible moves and must kick the cheating player from the game
- The server must wait for some reasonable amount of time (~30s) before starting a game. If the lobby does not fill in time (>= 4 players connect) the match is cancelled and connections are torn-down gracefully

4. Report

In addition to your server code you must write a report to discuss the following items.

- How would you scale up your solution to handle many more clients?
- How could you deal with identical messages arriving simultaneously on the same socket?
- With reference to your project, what are some of the key differences between designing network programs, and other programs you have developed?
- What are the limitations of your current implementation (e.g. scale, performance, complexity)?

In addition, if you are working as a group, report the contribution of each member of the team.

Your report should include your name(s) and student ID number(s). It should be no more than 2 pages.

5. Submission

Submit your **server code** and **report** on LMS by the deadline. If you are working in groups, only one member of the group should submit the assignment.

Code submission: You should submit a single code that captures all the tiers you have implemented. However, you may wish to submit separate codes for each tier (remember, higher tiers should satisfy all the functionalities of previous tiers).

Test environment: You may wish to specify the test environment if you find that a specific environment is needed to test your server (either Linux of Mac). Specify this clearly in the readme file, makefile, or at the top of your code (in the order of preference).

CITS 3002
Computer Networks

6. Additional resources

You may find socket programming tutorial useful to get started:
http://www.tutorialspoint.com/unix_sockets/socket_server_example.htm

CITS 3002
Computer Networks

7. Rubrics

	Criteria	Highly Satisfactory	Satisfactory	Unsatisfactory
Tier 1 (40 marks)	• Satisfy all conditions of tier 1 tasks • Game play with a single client • Code is reasonable to read with good comments	**Demonstrated the ability to implement all game features for a single player** • Correctly implement all functionalities for tier 1 tasks. • Code is clearly legible with sufficient comments to understand.	**Demonstrated the ability to implement some game features for a single player** • Many functionalities for tier 1 tasks are implemented. • Code is clearly legible with limited comments to understand.	**Failed to demonstrate the ability to implement a single player** • Very few functionalities for tier 1 tasks are implemented. • Code is not legible without sufficient comments to understand.
Tier 2 (15 marks)	• Satisfy all conditions of tier 2 tasks • Multiple (>4) players can play at the same time	**Demonstrated the ability to implement all game features for multi-players** • Players up to 4 can join the game. • Minimal performance overhead.	**Demonstrated the ability to implement some game features for multi-players** • Players up to 4 can join the game. • Some performance overhead.	**Failed to demonstrate the ability to implement multi-players** • Multi-players are not supported. • Massive performance overhead when playing.
Tier 3 (15 marks)	• Satisfy all conditions of tier 3 tasks • Handles player joining/exiting mid-game	**Demonstrated the ability to handle players joining/exiting the game** • Players can exit mid-game without interrupting the server. • Players must wait until current game finishes to join the game.	**Demonstrated the ability to handle players joining or exiting the game** • Players can exit mid-game without interrupting the server, or • Players must wait until current game finishes to join the game.	**Failed to demonstrate the ability to handle players joining/exiting the game** • Players exiting mid-game interrupts the server. • Players joining mid-game interrupts the server.
Tier 4 (10 marks)	• Satisfy all conditions of tier 4 tasks • Handles illegal moves • Wait for players to join game	**Demonstrated the ability to implement fair play** • Cheating players are distinguished and removed. • Wait appropriately for players to join the game.	**Demonstrated the ability to implement some fair play** • Some cheating players are distinguished and removed. • Has some waiting features for players to join the game.	**Failed to demonstrate the ability to implement fair play** • Cannot deal with cheating players. • Waiting for players to join is not reasonable or implemented.
Report (20 marks)	• Answer four key questions asked in section 4	**Demonstrated the ability to write a comprehensive report** • Detailed discussion for four questions asked. • Clear explanations and reasonable justifications are provided. • In-depth knowledge of the project demonstrated.	**Demonstrated the ability to write a reasonable report** • Some discussion for four questions asked. • Some explanations and reasonable justifications are provided. • Some knowledge of the project demonstrated.	**Failed to demonstrate the ability to write a report** • Failed to discuss the four questions asked. • Vague or no explanations and justifications are provided. • Not much knowledge of the project demonstrated.

This project assignment is out of total 100 possible marks.