Snakes

Department of Computer Science

1 Instructions

- This is **AN INDIVIDUAL** assignment. This assignment carries **60**% of the final **CPS2003** grade.
- While it is strongly recommended that you start working on the tasks as soon as the related material is covered in class, the firm submission deadline is **26th May 2017 at 17:00**. A hard copy of the report **is** required to be handed in.
- You are to allocate **30** hours for this assignment.
- A soft-copy of the report and all related files (including code) must be uploaded to the VLE by the indicated deadline. All files must be archived into a single .zip file. It is the student's responsibility to ensure that the uploaded zip file and all contents are valid.
- Reports (and code) that are difficult to follow due to low quality in the writing-style/organisation/presentations will be penalised.

This is the description of the assignment for unit CPS2003, Systems Programming, for the year 2017. This assignment is worth 60% of the total mark. The deadline of this project is 26th May 2017. This assignment must be attempted individually. Each individual might be asked to present his/her implementation, at which time your program will be executed and the design explained. Also, please read carefully the plagiarism guidelines on the ICT website.

2 Your Task

The aim of this assignment is to build a multi-player snakes game. It should be pretty easy to find out how to play this game[1]. For your implementation, I suggest you make use of the *ncurses* library, which allows fine control over a text terminal. Anyone running the command *snakes* will be placed in an "arena" as a snake 3 characters long. You will start in a position that is not occupied by any other player. Every half a second, the snake (and other players' snakes) will move 1 character forward in the direction it is facing (feel free to tweak the speed to make game playable). You can change the direction of movement by using the keys w, a, d and x. At random intervals, a special "fruit" will be placed in the arena. Anyone swallowing this fruit will become one character longer. If a snake hits the sides of the arena, itself or any other snake (assuming no other obstacles exits in your arena), the player will lose and the player's client program will terminate (note other players will continue playing). The player that reaches the length of 15 characters first will be considered the winner and the game will restart (together with all other players). The players might be joining the game either as a different user in the system or from a different system (over the network). Each player will see a current true picture of the arena drawn on his terminal. There should be no limit on the number of concurrent players allowed.

Your implementation will need to include two main programs: the server and the client. The server needs to be hosted on a machine which is accessible to all clients wanting to join the game. All requests by the client (such as moving the snake, creating a game etc...) should be performed by this server. It will also have the current configuration for any particular game (location of all the snakes, update rate, when and where fruit will appear, etc...) You can assume that only one game can be played at any one point in time. Clients located on different machines should communicate with the server through TCP/IP. Clients which are located on the same machine as the server can use alternate IPC mechanism for performance and efficiency. The server must be capable of handling multiple requests from different clients simultaneously.

3 Things to Consider

Marks will be allocated equally to a functional systems that allows multiple clients to access the server and perform operations on the game, as well as to good and efficient design. Several concurrent clients will be run on different machines. Having a system which is free of race conditions, deadlocks and still considerably efficient is ideal. Make sure that you spend enough time designing your system well before starting to code. Try

[1] https://en.wikipedia.org/wiki/Snake_(video_game)

to think of several scenarios which your game might encounter and design appropriate handling mechanisms. The following is a non-exhaustive list of things you might want to consider:

- How is a running game represented internally, and what data structures do I require?
- Are data structure instances unique to each client, or shared amongst multiple ones? Which one would be faster? What overheads are required to share data structures?
- What happens when two clients attempt to modify the same data structure concurrently?
- How are operations from multiple clients sent to the server?
- Is the server thread-safe?
- Is asynchronous functionality required?
- Are client requests queued and serialized, or is some form of parallelism implemented in the server? Does the latter option provide any benefits?
- What type of conflict resolution is required on the server?
- What happens when the connection between the server and client is lost?
- Are proper byte-ordering mechanism implemented on both the client and server?
- Would efficiency increase if clients cache the current map and only receive map updates from the server rather than the entire map every time?
- What happens when a player joins a running game (mid-game)?
- What happens if the server crashes, or the clients drop out mid-game?

You have to make sure that your system works on Linux-based systems running on i386 and x86-64 architectures. Both the client and server must be fault-tolerant, and can both be terminated with a CTRL-C or using the kill command. Ensure that proper clean-up is performed in each case. Thus each server must terminate all its child processes (if any) and close all open sockets. The child must close all its connections properly and make sure that any system-wide data structures are properly updated.

4 Deliverables

You must upload your source code (C files together with any accompanying headers), including any test clients and additional utilities, through VLE by the specified deadline. Use .zip or .tar.gz for archives. Include makefiles with your submission which can compile your system. These makefiles must generate the client and the server. Make sure that your code is properly commented and easily readable. Be careful with naming conventions and visual formatting. Marks may be deducted if I cannot understand what's going on in your code. Every system call output should be validated for errors and appropriate error messages should be reported. You must also hand-in a report describing:

- The design of your system, including any structure, behavior and interaction diagrams which might help.
- Description of any implemented mechanisms which you think deserve special mention.
- Details of the network protocols between the interacting processes, together with a diagrams visually describing these protocols.
- Your approach to testing the system, listing test cases and results.
- Server and client timings for above test cases, if you so desire.
- A list of bugs and issues of your implementation.

You do not need to describe in great detail your work, just make sure that anyone reading your report can understand the overall design concepts and how these were implemented. Please do not include any code listing in your report (snippets are allowed). Also, please use double-sided printing. You may submit your report, including the plagiarism form, to myself or the departmental secretary by the specified deadline.