The University of Queensland
School of Information Technology and Electrical Engineering

Semester One, 2010

COMP2303 / COMP7306 – Assignment 4.3

Due: 11:59pm Friday 4th June, 2010
Marks: 50
Weighting: 25% of your overall assignment mark (COMP2303)

Introduction

Your task is to write a multithreaded **TCP** server and client version of the naval game from Assignment 1. Your assignment must comply with the subject coding style using the C99 version of the language..

This is an **individual assignment**. You should feel free to discuss aspects of C programming and the assignment specification with your fellow students, but you shouldn't actively help (or seek help from) other students with the actual coding of your assignment solution. It is cheating to look at another student's code and it is cheating to allow your code to be seen or shared in printed or electronic form. You should note that all submitted code may be subject to automated checks for plagiarism and collusion. If we detect plagiarism or collusion, formal misconduct proceedings will be initiated against you. A likely penalty for a first offence would be a mark of 0 for the assignment. Don't risk it! If you're having trouble, seek help from a member of the teaching staff – don't be tempted to copy another student's code. You should read and understand the statements on student misconduct in the course profile and on the school website: http://www.itee.uq.edu.au/about_ITEE/policies/student-misconduct.html

Specification

You will produce two programs: `nserver` and `nclient`. The server will take four parameters.

1. the name of a logfile to record events in.
2. a strictly positive integer indicating the maximum number of simultaneous games which the server will support.
3. the name of the rules file to use.
4. the port the server should listen on. ~~This parameter is optional. If it is not given, the server should let the operating system choose the port.~~

Event	**Output**	**Exit status**
Incorrect number of params	Usage: nserver logfile max_games rules port	1
Invalid param types/values	Invalid param types or values.	2
Can't listen on port P.	Unable to listen on port.	3
Other network error	Network error.	5
Can't read rules file.	Error in rules file.	6

For this assignment, we will assume that the rules file is valid if it can be read.
A disconnection does not count as an error.
Server errors to go to **standard error**.

The server should play games until it is stopped (SIGINT). When the server exits it should flush its logfile. When the server starts it should print a message to both stdout and its log:

Server started on port X.

Where X is the port the server listens on.
If the server receives a SIGHUP signal it should output current player statistics to standard out. There will be one line per player with the following fields separated by a single tab:
playerid, games_won, games_lost, games_disconnected_from.
An early disconnection does not count as a win or loss for the other player.

The client takes four parameters.

1. a (non-empty) string identifying the player [no whitespace].
2. a (non-empty) string identifying the game to connect to [no whitespace].
3. the name of the map file for this player.
4. the port to connect to.

Note that the rules file will be read by the server, but the map will be read by the client.
Do not assume that the server can read the client map file directly.
If the server is doing most of the processing, then the client will need to send the contents of the map file to the server.
If the first player disconnects from the server before the second player, the first player will have a disconnect recorded against them and the game will be empty/removed.

You will not be marked on the output produced in Assignment 1. That is, the formatting etc of game messages is not part of the assessment. You do not even need to produce that output if you do not wish to. To make these programs easier to debug and mark, any output by the client which is assessable must begin with **I:** . Details of such messages will be explained later.

Server log
The server should record one line in its log for each event.

Event	**Log text**
Server starts	Server started on port *X*.
Server stops	Server stopped.
Client connects successfully	Client *ID* connected to game *G*.
Connection attempt to game which is full.	Rejected *ID* from full game *G*.
Connection attempt when max games running	Rejected *ID* due to too many games.
Game over – win	*ID* won game *G*.
Client disconnects before game is over	*ID* disconnected from game *G*.
Client disconnects due to bad map	ID disconnected due to bad map.

The clients should take turns with the first client to connect going first. The game ends when one of the clients sinks all of their oponent's ships.

Client messages
(Sent to standard out)

Event	**Output**	**Exit status**
Incorrect number of params	Usage: nclient id game map port	10
Invalid params types	I: Param error.	11
Map file is missing or unreadable	I: Missing map file.	30
Ships overlap in map	I: Overlap in map file.	50
Ship goes out of bounds	I: Out of bounds in map file.	51
Other error processing map file.	I: Error in map file.	52
Game over - win for me	I: Game over – win.	70
Game over – loss for me	I: Game over – loss.	71
Game over - disconnect	I: Game over – disconnect.	72
Unable to connect to server	I: Unable to connect to server.	80
Unexpected connection loss	I: Lost connection.	81
Program exits before end of game.		0

Interractions:

1. When the server determines that the game is over, it should send a message indicating why {win, loss, disconnect} and then disconnect.
2. If the client runs out of input before the game is over it should disconnect from the server.
3. If the first client disconnects from a game before a second client connects, then the game should go back to being empty.

Notes:

1. This assignment is to be multi-threaded; do not use `system()`, `fork()` or `select()` calls.
2. Be careful with your memory management. Your server's memory requirements should not grow significantly once it has reached the maximum number of simultaneous games.
3. Be careful to close unused files. Your server should not run out of file descriptors for a reasonable number of simultaneous games.
4. Only the server must be multi-threaded. The client can be if you wish but does not **need** to be.

Style:

You must follow **version 1.4** of the COMP2303/COMP7306 C programming style guide found at http://www.itee.uq.edu.au/~comp2303/resources/c_resources.html.
Note however that this assignment will be using the c99 form of the langauge (specifically -std=gnu99)

Submission:

Submission is via subversion. You should check out and commit to the **ass4/trunk** directory in your repository.

Your submission must include all source and any other required files (in particular you must submit a Makefile). Do not submit compiled files (eg .o or compiled programs).

Your Makefile must produce two programs called `nserver` and `nclient` respectively with the command:
```
make
```

Your program must be compiled under gcc with at least the following switches:
```
-Wall -std=gnu99 -pedantic
```

You are not permitted to disable warnings or use pragmas to hide them.
You are not permitted to import non-standard functions from standard libraries.
Your program must not use non-standard headers/libraries (except where they are written by you and submitted with your assignment).

The due date for this assignment is 11:59pm Friday 4th June, 2010. The policy on "grace days" and exceptional circumstances (e.g. illness) is as outlined in the course profile. Note that no submissions can be made more than 120 hours past the deadline under any circumstances.

Marks:

Marks will be awarded for both functionality and style.

Functionality (42 marks)

Provided your code compiles (see above), you will earn functionality marks based on the number of features your program correctly implements, as outlined below. Partial marks will be awarded for partially meeting the functionality requirements. Not all features are of equal difficulty. **If your program does not allow a feature to be tested then you will receive 0 marks for that feature**, even if you claim to have implemented it. The markers will make no alterations to your code (other than to remove code without academic merit).

Marks will be assigned in the following categories.

1. Server startup and argument processing (2 marks)
2. Play a single game (17 marks total)
   (a) client startup and argument processing
   (b) Correctly playing game and end of game messages
   (c) Statistics display on server
   (d) Server log file is correct.
3. Serial games (6 marks total)
   (a) Correct play and end of games.
   (b) Statistics display correct
   (c) Server log file is correct.
4. Concurrent games (17 marks total)
   (a) Correct play and end of games.

(b) Statistics display correct

(c) Server log file is correct.

Style (8 marks)

Your style mark will be the minimum of:

$8 * 0.9^{\text{(Number of style guide violations + number of compilation warnings)}}$
and
your functionality mark.

The number of compilation warnings will be the total number of distinct warning lines reported during the compilation process described above. The number of style guide violations refers to the number of violations of version 1.4 of the COMP2303/COMP7306 C Programming Style Guide. A maximum of 5 violations will be penalised for each broad guideline area. The broad guideline areas are Naming, Comments, Braces, Whitespace, Indentation, Line Length and Overall. For naming violations, the penalty will be one violation per offending name (not per use of the name) up to the maximum of five. You should pay particular attention to commenting so that others can understand your code. The marker's decision with respect to commenting violations is final – it is the marker who has to understand your code. To satisfy layout related guidelines, you may wish to consider the `indent(1)` tool. Your style mark can never be more than your functionality mark – this prevents the submission of well styled programs which don't meet at least a minimum level of required functionality.

Late Penalties

Late penalties will apply as outlined in the course profile if no grace days remain at the time of submission.

Specification Updates:

It is possible that this specification contains errors or inconsistencies or missing information. It is possible that clarifications will be issued via the course website. Any such clarifications posted 5 days (120 hours) or more before the due date will form part of the assignment specification. If you find any inconsistencies or omissions, please notify the teaching staff.

Tips/Notes:

1. It is up to you to decide what the responsibility of the client and server are (there is more than one correct way to do this).
2. Plan your threading out before coding. How many threads do you need on the client/server? What is each thread responsible for? Do you need to coordinate between threads?
3. Write some small test programs to use connections and threads to be sure you understand how to.
4. Neither your client nor your server should busy wait – non-trivial cpu usage while waiting may cost you marks.

Updates 4.1:

1. Added I: to the front of all client messages except for usage.
2. Parameters for the server and client must be as listed. Extra params are not allowed and should generate a usage error.
3. Fixed param order in usage error message.
4. Changed error message for invalid parameters on server.
   If the server cannot open the logfile for writing it should trigger this error.

Updates 4.2:

1. Fixed missing fullstop on one of the messages.
2. Removed port number from server error message to deal with case where port isn't known yet.
3. The validity of map files should be checked before clients are admitted to a game.
   That is a player with a bad map is not elligable to join games.
4. All non-network setup should be performed before opening network connections.
5. Be careful not to disconnect from the server until you are sure the game is over.
6. Added default exit status for client.
7. Failure to open the log file should be treated as a "bad param types of values".
8. If the rules file can be read it will be valid. That is, there will not be any formatting errors in it.
9. Server errors should be output to standard error.

Updates 4.3:

1. Fixed the due day.
2. If a client connects with an invalid map the server should now create a log entry to that effect.
3. The port argument on the server is not optional any more.
4. Fixed inconsistent full stop.
5. If the server cannot write to its log file it should exit with error 2.