COP5570 Project No. 3: Online tic-tac-toe Server

PURPOSE

- Experience with socket programming, experience with implementing a network application.

DESCRIPTION

You can do this assignment in a group of two. In this assignment, you will implement a game server for the online tic-tac-toe game. The tic-tac-toe game is played on a 3×3 board with two types of stones, black and white. Each player takes turn to play (place a stone anywhere in the board that is still available), with the black always making the first move. The game is won by having three consecutive stones horizontally, vertically, or diagonally on the board. The server should support the basic functionality of a typical Internet game server:

- A new user can login as a guest and register username and password for the server. After a user registers, (s)he can login with the username and password.
- A user may play games with other users, observe games currently being played, and comment on the games observed.
- The server monitors the progress of each game, deciding whether the game finishes and checking the time used by each player.
- The server provides basic statistics of the strength of each user such as win/loss ratio and rating.
- The server provides communication support among users. A user should be able to carry out one-to-one communication with another user, broadcast messages to all current online users, and leave a message when a user is on-line or not on-line. A user may choose not to receive any broadcast messages (quiet mode) or to block everything from particular users.
- The server should provide some utility functions such as listing all online users, displaying the information about the games and users.

More specifically, the following commands should be supported in our tic-tac-toe server:

```
[...] optional field, <......> required field

who                     # List all online users
stats [name]            # Display user information
game                    # list all current games
observe <game_num>      # Observe a game
unobserve               # Unobserve a game
match <name> <b|w> [t]  # Try to start a game
<A|B|C><1|2|3>          # Make a move in a game
resign                  # Resign a game
refresh                 # Refresh a game
shout <msg>             # shout <msg> to every one online
tell <name> <msg>       # tell user <name> message
kibitz <msg>            # Comment on a game when observing
' <msg>                 # Comment on a game
quiet                   # Quiet mode, no broadcast messages
nonquiet                # Non-quiet mode
```

```
block <id>              # No more communication from <id>
unblock <id>            # Allow communication from <id>
listmail                # List the header of the mails
readmail <msg_num>      # Read the particular mail
deletemail <msg_num>    # Delete the particular mail
mail <id> <title>       # Send id a mail
info <msg>              # change your information to <msg>
passwd <new>            # change password
exit                    # quit the system
quit                    # quit the system
help                    # print this message
?                       # print this message
register <name> <pwd>   # register a new user
```

1. Anyone can login using the guest account for registering.
2. Register should only be allowed in the guest account.
3. Account information should be retained even if the server crashes. The account information should be stored in a sufficient frequency (e.g. every 5 mins or every time a game is played, etc).
4. The first match command should send a message to the other player indicating that somebody is inviting him to play a game. The second match (from the other player) may start the game when everything agrees.
5. After a game starts, the player types in the coordinate to place each stone. In this game, the coordinate is represented as XY, where $X \in [\prime A\prime, \prime B\prime, \prime C\prime]$ and $Y \in [1, 2, 3]$.
6. The server must keep track of the time used by the two players in a game. The one who uses up all his/her time will lose by time.
7. The serve must check if either player wins the game after every move.
8. When a user in a game disconnects, (s)he loses by default.
9. The server should keep win/lose records for all users and maintain some kind of rating system.
10. A kibitz message should only send to the ones observing the same game.
11. Messages can be left for a user on-line or off-line.
12. The messages for a user must be marked as either read or unread when listed.
13. When a user login, there must be a message indicating the number of unread messages in his mailbox.
14. Messages can be any reasonable sized (e.g. $<$ 1MB) multi-line text messages.
15. A player in the quiet mode does not receive any broadcast messages (from shout and kibitz), but should receive personal message (tell and mail).
16. A player may block any communication (should, kibitz, tell, and mail) from another player by putting the player in his/her blocked list. You can have a limit ($>= 5$) on the number of players to be blocked.
17. Quiet, nonquiet, block, and unblock commands should have effect across logins.
18. When a mail is sent to a user that is on-line, a message should also be sent to the user (notifying the incoming message).

19. **The server should not be affected in anyway by the user behavior.** For example, no matter what a particular user does, the server needs to be able to serve other users.
20. The server should work with the plain telnet client.
21. You can assume the maximum number of players online to have a reasonable limit (e.g. 20).

A sample functional but perhaps buggy executable is running on linprog6 ports 55555, 56666, 57777, 58888, 59999 (e.g. 'telnet linprog6 58888' from linprog1 to use the sample game server). You should try to make your server behave like the sample server **without bugs**.

DEADLINE AND MATERIALS TO BE HANDED IN

Deadline: March 29.

- Submit all files related to the assignment including makefile, README file, and your self-grading sheet in one tar file to canvas.

  In a project directory that contains your submitted files, a 'make' command should create the executable in linprog. Your README file should describe how to compile and run you program, the known bugs in your program, and how to do your demo. The makefile should (1) automatically generate the executable by issuing a 'make' command under the directory, (2) compile the C/C++ files with '-Wall -ansi -pedantic' or '-Wall -std=cxx -pedantic', (3) clean the directory by issuing 'make clean', and (4) recompile only the related files when a file is modified.

GRADING

The program only needs to run on linprog. You can only use C or C++ for the assignment. You must use system calls such as `socket`, `bind`, `accept`, `select`, `write`, and `read`, `open`, etc, to deal with communication and files. You are not allowed to use any higher level communication and/or process/thread libraries. Violating this will receive a 0 point grade. A program with compiler error will receive a 0 point grade. A program that does not provide game server functionality will receive 0 point.

1. `register` (5)
2. `exit, quit` (2)
3. `tell` (5)
4. `shout` (5)
5. `help, ?` (2)
6. `who` (5)
7. `stats` (5)
8. `block, unblock` (5)
9. `quiet, nonquiet` (5)
10. `listmail` (5)
11. `mail/readmail` (5)
12. `deletemail` (5)

13. `match` (5)
14. `observe, unobserve` (5)
15. `resign` (5)
16. `game` (5)
17. `kibitz (’), refresh` (5)
18. game play (move, time) (10)
19. `info, passwd` (5)
20. overall (disconnect, everything combined) (6)
21. demo and project submission (5)
22. 30 extra points for a window based client for the server (you can use any language or package for this task).
23. The project will receive at most 85% of points with the third unknown bug.
24. extra -8 for the third known/unknown bug and unimplemented feature.
25. -5 for each compiler warning (with the required flags).

Misc.

You should start from an echo server to build up the game server. The sample program is a multiplex server (2000+ lines of code).

You should focus on the correct user behavior. You will notice many unspecified features in the project (e.g. the program behavior when a second connection from the same user). Your server needs have a reasonable behavior in all cases. Under no circumstances should the server crash.

All programs will be checked by an automatic software plagiarism detection tool.