CS 4119 : Computer Networks
Programming Assignment I

Spring 2013

Due: Tuesday, March 5th

Academic Honesty Policy

You are permitted and encouraged to help each other through Piazza's web board, in order to discuss and understand concepts learned in class or go deeper in a particular subject. **HOWEVER**, you may **NOT** share source code or hardcopies of source code, or answers to written assignments. Refrain from activities or the sharing materials that could cause your source code to **APPEAR TO BE** similar to another student's enrolled in this or previous years. Cheating will be dealt with severely. Cheaters will be punished. Source code and written answers should be yours and yours only. Do not cheat.

1 Introduction

In this assignment, you will work on the application layer over UDP. The goal is to implement a system comprising of a game server and clients. The game in question is tic-tac-toe.

The assignment can be broadly divided into 3 parts. In the 1st part, you will be designing a system with a server and clients. The clients can basically login to the server, and play with other clients using the server. In the 2nd part, the clients-server system will be passed through an unreliable channel (which is provided by us) so that packets are dropped from the client to the server **NOT** vice-versa. This is to test whether the acknowledgement protocol devised by you is robust enough to handle packet losses. The last part which is not connected to the 2nd part is to use a proxy server (also provided by us) to connect from the client to the game server.

There are certain restrictions w.r.t. the internal implementations and data format all of which you have to adhere to strictly.

2 Due Dates and Program Submission

This programming assignment is due on **March 5th**. **No lateness is accepted**.

Before submitting, make sure your program compiles and runs properly on a CS or EE machine, because you will be graded on those machines and if your code doesn't compile, you automatically get a grade of 0.

To submit the assignment, rename the **yourUNI** folder provided with the sample code to your UNI (eg. sb3457). Zip this folder with the same name, and upload this zipped file on CourseWorks.

Ensure that you provide a MakeFile along with your assignment for us to compile.

3 Getting started

Name the file containing the main function of client as 'client' and server as 'server'.

3.1 Introducing the System

This section will introduce the functionalities of the client and the server, including the naming convention and arguments required for each of the client and server programs.

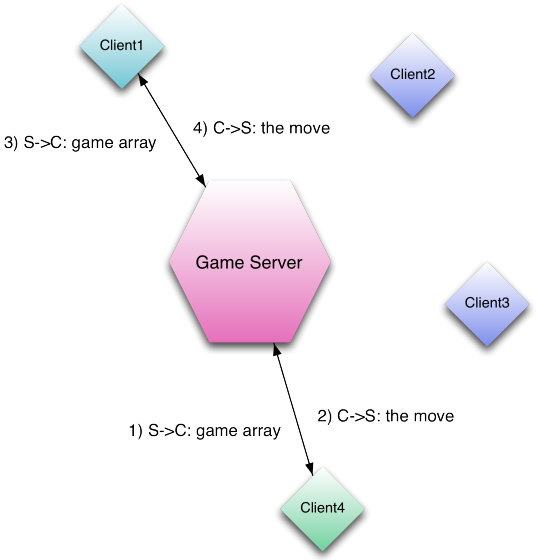


Fig 1 shows the "Login" module.

The Client has the following functionalities :-

- Login to the server - one-time registering of the client at the server
- Query the List - the client can at any time when it's "free" query for a list of other clients registered on the server
- Choose a player - it can choose to initiate a connection with another client to play the game
- Play the game - choose a cell to play
- Logout of the server - terminate connection with the server
- Resending 'lost' packets - resend packets which have been lost over an unreliable channel

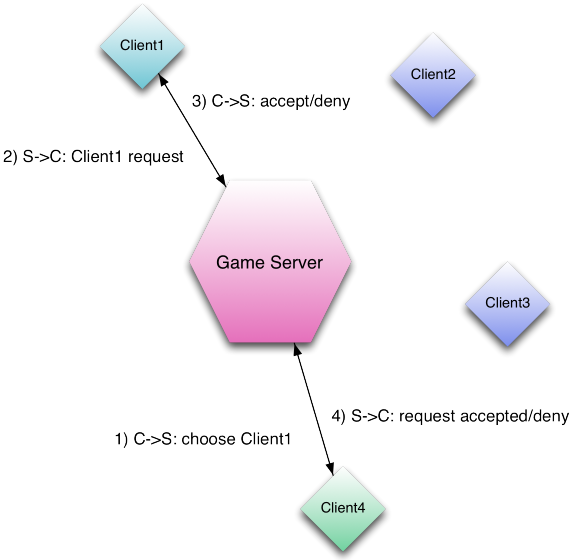


Fig 2 shows the "Game" module.

The Server meanwhile concentrates on :-

- Maintaining a list of clients - must be in alphabetical order
- Handling game logic - decide if a current move by the client is valid, decide whether the game is over or not, decide whose turn it is to play
- Pairing up clients - ensure that if a client sends a request to play with another client, send a failed request (ackchoose,F) to all other clients who want to play with either of these clients ; once someone accepts or denies, send the appropriate message to each client
- Sending acknowledgement packets - send ACK packets to the clients so that the receipt of the packets is confirmed

Apart from handling the functionalities, the following constraints are also placed on the information/packets that are sent between the client and the server. This format is valid for all parts.

The client has to type commands in the terminal. The following table lists the information at the Client side:-

Command in the following table is what is typed in the terminal.
Packet Content in the following table is the payload of the UDP packet sent from client to server.

Functionality	**Command**	**Packet Content**	**Packet Content - Example**
Login	login <name>	login**,**<packetId>**,**<name>,<port>	login,1,john,2134
Query List	ls	list**,**<packetId>,<name>	list,2,john
Choose Player	choose <name2>	choose,<packetId>,<name>,<name2>	choose,3,john,matt
Accept Request	accept <name1>	ackchoose,<packetId>,<name>,<name1>,A	ackchoose,2,matt,john, A
Deny Request	deny <name1>	ackchoose,<packetId>,<name>,<name1>,D	ackchoose, 3,matt,john,D
Play Game	play <number>	play,<packetId>,<name>,<number>	play,2,john,9
Logout	logout	logout,<packetId>,<name>	logout,john

Similarly the following table lists the information at the Server side:-
Packet Content in the following table is the payload of the UDP packet sent from server to client.

Functionality	**Meaning**	**Packet Content**
Login Acknowledgement	fail if another client with the same name exists	acklogin, [F/S]
Return the list	sends to the client the list of other clients currently registered	ackls, [$name_i$, $state_i$....]
Request a client chosen	sends a play request to a client from another client	request, <name>
Inform the client of the choice outcome	informs the client whether his request has been accepted/denied or has failed	ackchoose,<name>,[A/D/F]
Acknowledgement on receiving packets	sends acknowledgement to the client on receiving a packet	ack,<packetId>
Show the current state of the game during client's turn	shows the current state of the grid	play,<game_state>
If the current move is not allowed	checks if the query by the client to the server is a valid one	ackplay,[O/T]
Display result once the game is over	display whether the game has been won/ lost or drawn	result,[W/L/D]

<name> - refers to either characters or digits with no whitespaces, is the name you give at login
<name1> - is the client who sends the request
<name2> - is the client who is chosen (who will receive the request and make "accept/deny" decision)

<packetId> - is a unique Id maintained by each client for each packet that it sends (maintain a counter variable and increment it as you assign it to a new packet)
<number> - stands for the 1 digit number stands for the location to play
[F/S] - stands for Failure/ Success, either one of the states

[$name_i$, $state_i$.....] - a comma separate list containing the name and state of other clients currently registered in the server
note: state can be of three types [free/busy/decision]. A client is free when it has neither sent a request to another client nor has currently recieved a request from another client in which case its state is 'decision'. If a client has accepted any request from another client or has a request accepted, it's state would be set to 'busy, if a client denies a request, set it's state and the state of the client that sent the request to 'free'. Set the state of the client to 'free' once a game is over between a pair of clients. Also ensure that a choose request can be sent by or sent to a 'free' client only.

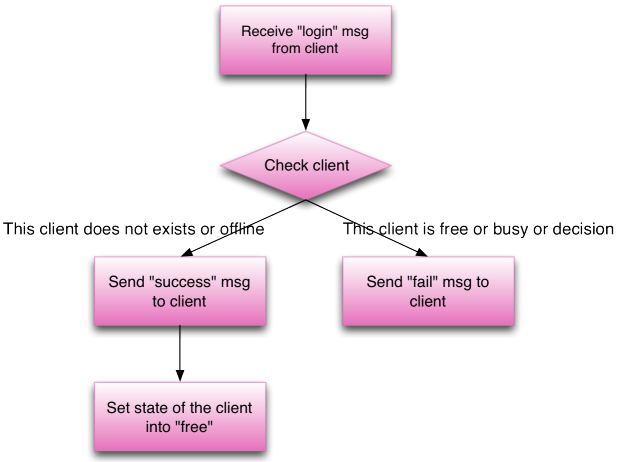
[A/D/F] - Accept/Deny/Failure, either one of the states
<game_state> - is a 9 digit string, initially all 0's (indicate unoccupied place), replaced with 1's and 2's standing for each player
[W/L/D] - stands for Win, Loss and Draw, either one of the states
[O/T]- stands for Occupied, (Out-of-Turn), either one of the states

The <number> stands for a location on the 3*3 tic-tac-toe grid which stands as follows :-

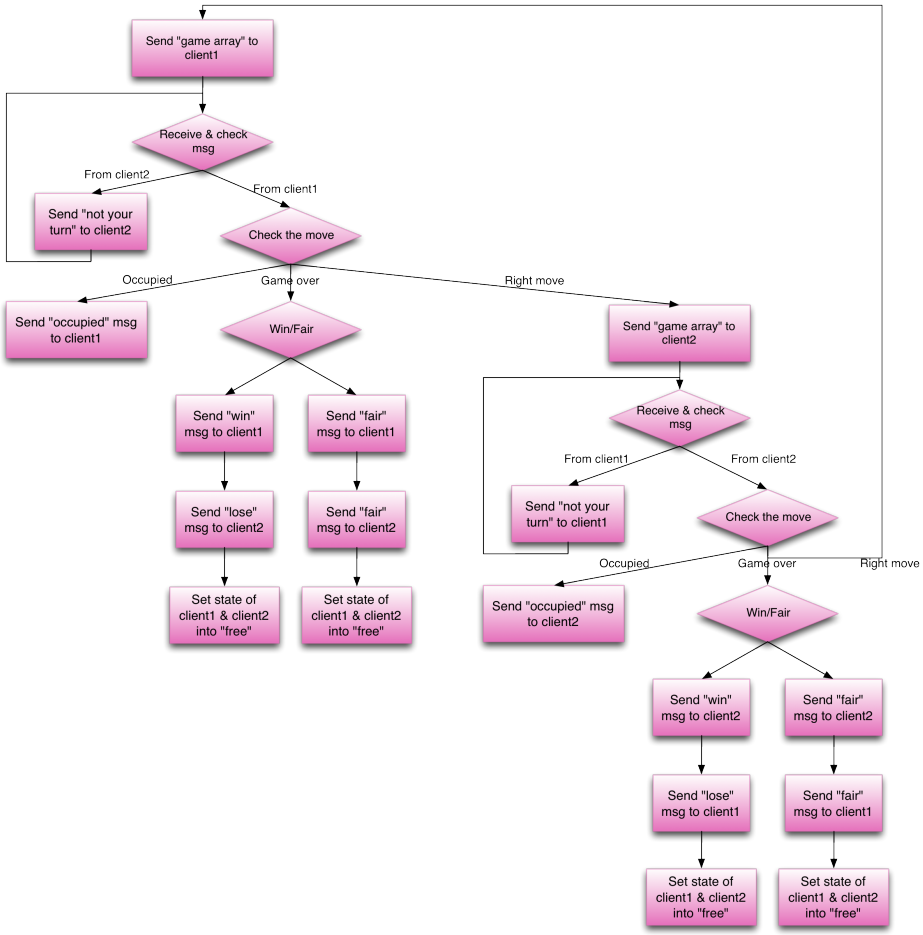
1	2	3
4	5	6
7	8	9

3.2 Flowcharts

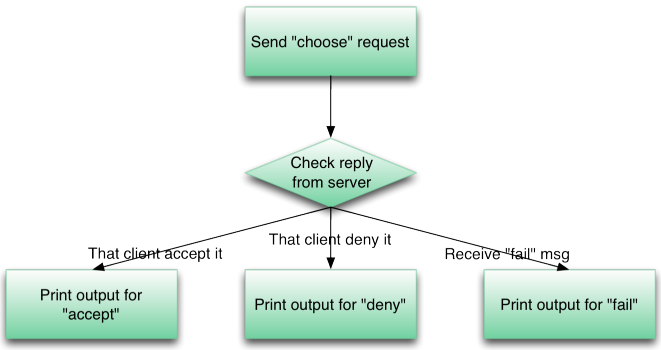
3.2.1 Login (Server)



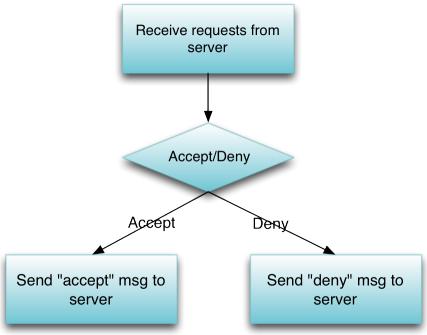
3.2.2 Game (Server)



3.2.3 Request Choose (Client)



3.2.4 Decision Choose (Client)



3.3 The environment

Your program will be run by instantiating multiple client programs and a single server program on different terminals on the same machine. So the IP address would be the same for all parties involved in this environment so we treat their port numbers (i.e. the port which each entity (client/server) listens to) as a way of distinguishing one from the other. Fix the server's listening port as 4119. This information is available to the clients before runtime.

To instantiate clients for example in java, type - **java client 1234 127.0.0.1 4119** where 1234 is the port which the client listens to, 127.0.0.1 is the IP of the machine in which the server runs (in this case the same machine) and 4119 is the port which the server listens to. Please make sure no two clients listen at the same port. To instantiate server for example in java, type - **java server**.

4 The Assignment

4.1 Part 1

This part basically tests if your system comprising of the clients and the server works as intended.

4.2 Part 2

In this part, the client sends to the server via a channel but not vice versa. We provide the class files for the channel as a jar file. Instantiate the channel by typing **java -jar channel.jar 4312** where 4312 is now the port.

While running this part reinstantiate the clients using the new port number 4312 instead of 4119.

4.3 Part 3

In this part, we enforce the constraint that the server cannot accept more than 5 clients playing from the same IP. Hence the 6th client from the same IP is blocked by the server. So you can bypass this by letting this client access the server via a Proxy Server. We provide the class files for the proxy server in a jar file. Instantiate the proxy server in a different machine by typing **java -jar proxy.jar 191.148.1.2 4119 5000** where 191.148.1.2 is the IP to which it sends the message (here the server's IP), 4119 is the port number which the server listens to, and 5000 is the port which the proxy server listens to.

As in the 2nd part, reinstantiate the clients using the new IP and port number of the proxy server.

5 Output

Please print on the screen the following as and when certain events occurs. The events are specified in the comments '//' (do not print comments).

Pay attention not to add extra whitespace in your output, just follow the format.
Do not print '<' or '>' in your output, that is a way to show the format.
All printing is at the client side only.

// Print once the server acknowledges your login
// Login:
login success <player name>

// Print once the server responds to your 'ls' query, in alphabetical order by name
//List:
<player name1> state1
<player name2> state2
<player name3> state3
EOL

// Print once of these actions are taken by any client
// Choose:
request from <player name1>
request accepted by <player name2>
request denied by <player name2>
request to <player name2> failed

// Print once you receive multiple choose requests from the same client Only one request
// Print once the client receives the current configuration from the server
//Play:
- 1 -
- - 2
1 - -

//Print once the client chooses the cell to play at
<player name> number

//Print the contents of all ackplay and ackchoose packets when the client recieves them

// Print once the game ends
// Game end:
<player name> win/lose/draw

// Print once the player logs out
// Logout:
<player name> logout