COMP 3331/9331 Assignment T3 2021

Instant Messaging Application

All details are in the specification

- **READ THE SPECIFICATION**
- **READ THE SPECIFICATION (AGAIN)**
- Information about deadlines, file names, submission instructions, marking guidelines, example interactions and various other specifics are in the specification
- Choice of programming languages: C, Java, Python (versions are noted in the specification)
- This talk provides a high-level overview

Two main components

- Server
  - Always on
  - Does not remember state from prior executions, on start-up no clients are initially active
  - Responsible for
    - User authentication
    - Direct messages between clients (online or offline)
    - Additional functions: presence notification, blacklisting, timeout inactive users
    - Participates in setting up a p2p messaging session between two clients
  - Implements a request/response API for interacting with client(s)
- Client
  - Interacts with user through command line
  - Exchanges messages (request/response API) with the server to manifest the commands
- Transport Protocol: TCP (required)
- You must design your own **application layer protocol**
  - This includes the syntax/semantics of the messages exchanged between the client and server and the actions to be taken upon receiving each message

Execution

- Client and server executed on same machine
  - Assume IP address of the other endpoint is 127.0.0.1 (local host)
- Server
  - Command line arguments:
    - Server port (use a value greater than 1023 and less than 65536)
    - Block duration in seconds (the duration for which a user is blocked after 3 unsuccessful login attempts)
    - Timeout in seconds (the duration of inactivity after which a user is logged off)
  - Executed first – waits for client(s) to connect
- Client
  - Command line arguments:
    - Server port number (should match the first argument for the server)
  - Let the OS pick an an available port
  - Client should initiate TCP connection with server (“127.0.0.1”, server port)
  - User should interact with the client through the command line (command prompt, console)

Part 1: Client-Server Mode

- **User Authentication**
  - Credentials file will be available in current working directory of server with read and write permissions set (chmod +wr) – sample provided
  - Ask user to enter username (assume username is in correct format)
  - If username exists and the user is not already logged on (from another client), then prompt user for password
    - If password matches, the user is assumed to be logged on
    - If password doesn't match, aske user to enter password again
    - If a user successively fails three times, then they are blocked for the block duration – i.e., the user cannot login again for that time
  - If username doesn't exist prompt the user for a new password
    - Create a new entry in the credentials file (assume password is in correct format, no checks necessary)
- **Presence Broadcast**
  - When a user logs on/off, display a message to all online users excluding those who this user may have blocked
- **Detection of inactivity**
  - If a user does not issue a valid command (client-server or p2p) for **timeout** seconds, they are assumed to be inactive and logged off

Part 1: Client-Server Mode

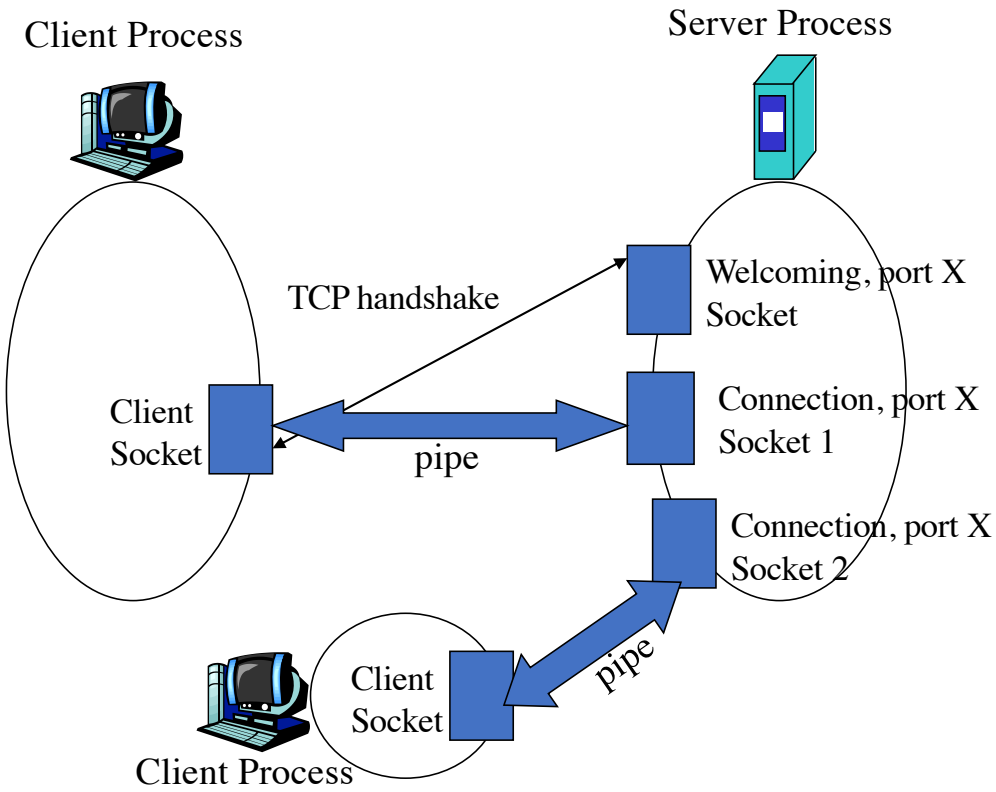
- **message user**
  - Assume message is in the correct format (no checks are necessary)
  - If user is online and has not blocked the sender, send the message
  - If user is offline and has not blocked the sender, store the message for delivery when user logs on
  - If user has blocked the sender or is invalid (i.e., username does not exist in the credentials file), then report an appropriate error message
- **broadcast message**
  - Send message to all online users except those who have blocked the sender (in which case, report to sender that message could not be sent to some users)

Part 1: Client-Server Mode

- **whoelse**
  - Display names of all users currently online excluding those who have blocked the user executing the command
- **whoelsesince time**
  - Display names of all users who may logged in at any time within the past **time** excluding those who have blocked the user executing the command
- **block user**
  - blocks user from sending messages to command issuer, receive presence notifications about command issuer, and check online status (i.e.,whoelse, whoelsesince) about command issuer
- **unblock user**
  - reverse of above
- **logout**
  - Log out user and send presence notification and close a p2p messaging session if active
- Handle simple errors
  - E.g., invalid commands or blocking an invalid user, etc
  - Simple error message displayed to user

Part 1: Client-Server Mode

- Server should support multiple clients
- All clients and server executing on same machine (127.0.0.1)
- Multi-threading
  - Main thread waits for a new TCP connection, creates the connection socket and spawns a child thread for interacting with one client



Part 2: Peer to Peer Messaging

- **Setup process managed by server**
  - User A will initiate **startprivate** command with user B
  - Server will check if user B has blocked User A
  - Server will check with user B if they are willing to accept
  - If OK to proceed, then certain information about B should be conveyed to A so that A can initiate a TCP connection with B
- **Private user message**
  - Once p2p session is setup, the two users can message each other directly bypassing the server
- **Stop private**
  - Ends the p2p session
- Each user can be involved in at most one p2p session at any given time

Part 2: Peer to Peer Messaging

- Possible for user to move back and forth between client/server and p2p messaging commands
- Client needs to maintain a TCP connection with another client in addition to maintaining the TCP connection with the server
- Client also needs to interact with the user through command-line (even in client/server interactions) while also simultaneously interacting with the server and other client (in p2p messaging)
- Both above can be readily achieved via multi-threading. Alternately, you may explore non-blocking IO (select)

Data Structures

- Server must maintain certain state information including # of valid/online users, login times, blocked user lists, offline messages, etc.
- Proper design of data structures is essential to ensure all functionality can be achieved
- Be careful about accessing the data structures across multiple threads
- Avoid arbitrary upper limits for variables (e.g., no of users)
  - Dynamic memory allocation is recommended

How to start?

- How to get started
  - Start with a server supports only one client at a time
  - Add user authentication
  - Extend server to support multiple client
  - Add presence notification
  - Add messaging (1 to 1, broadcast, offline)
  - Add inactivity detection
  - Add support for checking online (+history) queries
  - Add blocking
  - Next move to p2p messaging

Messages displayed to users

- Use meaningful text
- Does NOT have to match the examples in the specification
- Assignment will be **MANUALLY** marked by your tutors
- No messages at the server

Report

- Program design
- Data structures
- Details of the application layer protocol
- Trade-offs considered
- Point out issues if program does not work under certain circumstances
- Refer to all borrowed code

Testing

- Test, Test, Test
- Server and client(s) executing on **same machine**
- Emphasis on correct behaviour
- Basic error checking
- **MUST test In VLAB environment and through command line**
- If we cannot run your code, then we cannot award you any marks
- Detailed marking rubric is available in specification
- SYSTEMATIC DEBUGGING + UNIT FUNCTIONS

Plagiarism

- **DO NOT DO IT**
- **Both parties involved are considered guilty**
- **If posting code on online repositories, then ensure access is private**
- If caught
  - You will receive zero marks (and there may be further repercussions if this is not your first offence)
  - Your name will be added to the school plagiarism register
- Every term I have had to do the above for a small group of students
  - Would be nice to not have to do it in this term

Non-CSE Students

- Must be enrolled in a non-CSE program (double degree that includes a CSE major does not qualify)
- Can opt for non-CSE option
- MUST request permission to do so by 5pm, 15th October by emailing cs3331@cse.unsw.edu.au

Resources

- Many program snippets are on the web page
  - Including multi-threading code snippets
- Your socket programming experience in Labs 2 and 3 will be useful
- Repository of resources is here

https://webcms3.cse.unsw.edu.au/COMP3331/21T3/resources/65959

Seeking Help

- Assignment specific consults (for all 3 programming languages) from Weeks 7-10
  - Schedule to be announced in a few days
- Course message forum
  - Read posts from other students before posting your question
- Read the spec – very often your answer will be in there