Computer Network 2010
Final Project

Intro to the Baseline Project

Multi-user Multi-channel Chatroom

- Something like IRC (Internet Relay Chat)
- Something like Chatroom function in PTT
  - Main menu→ (T)alk → (C)hat
- You will need to implement both the server and the client
  - In C/C++/Java, plz avoid using RAD components
  - You can use any protocol, but should be reliable

Login

- The server must be able to serve many users at the same time
  - However, you can assume the server capacity is 128 users
- After the connection established, the server will ask for the user's nickname. No space allowed in the nickname.
  - E.g. LOGIN:> [nickname]
- After the user login into the system, they will be assign a default channel and tell others its join
  - The prompt format is '#[channel name]:>
  - E.g. when A login, B will see
    - #default:>
    - ** User A join the default channel
    - #default:> _

Chat

- 1. What the user said will be send to all others in the same channel
  - User A:
    - #default:> Hi all~
  - User B:
    - #default:>
    - A: Hi all~
    - #default:> _
- 2. The user can use specified commend to send/reply private message via the server to others
  - User A:
    - #default:> /tell B Hi all~
  - User B:
    - #default:>
    - A tells you: Hi all~
    - #default:> _

Chat

- 3. Users can ignore/unignore specified user's message
  - User B:
    - #default:> /ignore A
    - ** All message from A will be discarded
  - User A:
    - #default:> /tell B Hi~
  - User B:
    - #default:> _ [nothing will happen]
  - User B:
    - #default:> /unignore A
    - ** All message from A will be shown
  - User A:
    - #default:> /tell B Hi~
  - User B:
    - #default:>
    - A tells you: Hi~
    - #default:> _

Multi-channel Support*

- Users will be able to join/create new channels
  - The channels will be destroyed if no user in it
  - If the channel the user wants to join doesn't exist, the system will create it automatically
  - User A:
    - #default:> /join TEST
    - ** A join TEST
    - #TEST:> _
- Users will be able list all current channels and its user count
  - User A:
    - #default:> /channels
    - ** default 1
    - ** TEST 2
    - #default:> _

* This is a bonus part, so you can decide if you want to implement it

List users

- List who is on-line or in specified channel
  - User A:
    - #default:> /listall
    - ** A in default
    - ** B in TEST
    - ** C in TEST
  - User A:
    - #default:> /list TEST
    - ** B in TEST
    - ** C in TEST

File Transfer

- The data communication channel are not required to separate from the one you already established, but if you separate them, you will get extra bonus
- If you do not implement the multi-channel support, or you don't know how to store files for each channel, you can just put those files in a global pool.

- Users can upload file to a channel
  - #default:> /up 123.mov
  - ** Upload 123.mov to default channel
  - ** ClientPort[21452] --> ServerPort[52320]
  - ** Upload completed!
- Users can list the files stored in the channel
  - #default:> /listfile
  - ** 123.mov 61118813 ← this is the file size
- Users can download the files stored in the channel
  - #default:> /dl 123.mov
  - ** Downloading 123.mov from default channel
  - ** ClientPort[21478] <-- ServerPort[52324]
  - ** Download completed!

Private File Transfer

- When A wants to send a file to B, A will ask if B wants to receive the file and then upload it to the server. B will automatically download the file after the upload is completed. The file will not be listed by anyone and will be deleted after the download completed.
  - User A:
    - #default:> /sendfile B 123.mov
    - ** Asking B for permission to send 123.mov
    - #default:> _
  - User B:
    - #default:>
    - ** A sends you 123.mov size=61118813, receive it? [y/n] y ← B need to reply the request
    - ** Waiting for 123.mov
    - #default:> _
  - User A:
    - #default:>
    - ** Upload 123.mov to B
    - ** ClientPort[21586] --> ServerPort[43583]
    - ** Upload completed!
    - #default:> _
  - User B:
    - #default:> _
    - ** Downloading 123.mov from A
    - ** ClientPort[51620] <-- ServerPort[43623]
    - ** Download completed!

*A small bonus here:

Send a file to multiple users at once

User A:

#default:> /sendfile B C 123.mov
** Asking B for permission to send 123.mov
** Asking C for permission to send 123.mov
#default:> _

Logout

- Exit the chatroom system, but you must be able to tolerate the abnormal disconnect
  - User A:
    - #default:> /quit
    - ** You are disconnected
- If the user A in default channel disconnect
  - User B:
    - #default:> _
    - ** A disconnected
    - #default:> _

Server-side Debug Messages

- For those functions you do not implement, just **skip them**
- Every 30 seconds print out:
  - 1. How many users on-line(have nickname) and its channel
  - 2. How many connection slots are in use including used by other functions

    ** 2 users online, 3 connection slots are in use @ [time]

    A default

    B TEST
- Print out when…
  - A user join the chatroom system
    - [user] login from [IP] @ [time]
  - A user leave the chatroom system
    - [user] logout @ [time]
  - A channel is created
    - channel [channel] is created by [user] @ [time]
  - A channel is destroyed
    - channel [channel] is destroyed by [user] @ [time]

Server-side Debug Messages

- Print out when…
  - A file is uploaded to a channel
    - FILE: [filename] in [channel] upload completed @ [time]
  - A file is deleted from a channel
    - Note that when a channel is destroyed, all its file will be deleted
    - FILE: [filename] in [channel] deleted @ [time]
  - A file transfer to a user started
    - FILE: [filename] in [channel] send to [user] started @ [time]
  - A file transfer to a user ended
    - FILE: [filename] in [channel] send to [user:port] completed @ [time]
  - A private file transfer from a user to another started
    - FILE: [filename] from [user] send to [users] is uploading @ [time]
  - A private file transfer from a user to another ended
    - FILE: [filename] from [user] send to [users] completed @ [time]

Directly Client-to-Client File Transfer*

- Send a file directly from a user to another
- The client software sometimes will act as a server to receive the file from others
  - User A:
    - #default:> /dccfile B 123.mov
    - ** Asking B for permission to send 123.mov
    - #default:> _
  - User B:
    - #default:>
    - ** A directly sends you 123.mov size=61118813, receive it? [y/n] y
    - ** Start to receive 123.mov from A[114.42.165.116:37260]
    - ** 123.mov from A[114.42.165.116:37260] is completely received
    - #default:> _
  - User A:
    - #default:>
    - ** Directly sends 123.mov to B
    - ** ClientPort[37260] --> [140.112.29.100:27284]
    - ** Upload completed!
    - #default:> _

* This is a bonus part, so you can decide if you want to implement it

On-the-fly Private File Transfer*

- Send a file from a user to another via the server on-the-fly
- The server doesn't store the file temporarily, but act as a proxy between those two users
  - User A:
    - #default:> /proxyfile B 123.mov
    - ** Asking B for permission to send 123.mov
    - #default:> _
  - User B:
    - #default:>
    - ** A sends you 123.mov via the server size=61118813, receive it? [y/n] y
    - ** Start to receive 123.mov from A via server
    - ** ClientPort[27690] --> ServerPort[61254]
    - ** 123.mov from A is completely received
    - #default:> _
  - User A:
    - #default:>
    - ** Sends 123.mov to B via server
    - ** ClientPort[42318] --> ServerPort[61256]
    - ** Upload completed!
    - #default:> _

* This is a bonus part, so you can decide if you want to implement it

Additional Server-side Messages

- If you don't implement the two extra parts above, you don't need to show those messages, too
- Print out when…
  - A directly client-to-client file transfer
    - * FILE: [filename] from [user] dcc-send to [user] @ [time]
  - A on-the-fly private file transfer from a user to another started
    - uport: user's port, sport: server's port
    - * FILE: [filename] from [user]:[uport]->[sport] proxy-send to [sport]->[user]:[uport] started @ [time]
  - A on-the-fly private file transfer from a user to another ended
    - * FILE: [filename] from [user]:[uport]->[sport] proxy-send to [sport]->[user]:[uport] completed @ [time]

What You Need to Know Before Start?

- Socket programming slide may provide some information
- Basic Socket API
  - How to write a TCP server?
  - How to write a TCP client?
- Construct a concurrent server
  - Multi-threading + Mutex
  - Multi-process + Share Memory + Semaphore
  - Single process I/O multiplexing

Questions?

Hints

- Multi-channel Support
  - You may need to store information of clients which contains their channel
- Private File Transfer
  - Just think you need to ask many people if they want to receive the file, and keep the file until those who want to receive it completely download the file

Hints

- Directly Client-to-Client File Transfer
  - The clients sometimes need to act as a server and sometimes may need to establish an additional connection for the transfer
- On-the-fly Private File Transfer
  - The server need to act as a proxy, so you will need to maintain this until they finish the transfer

Before you come to demo…

- Write a simple report (text file is fine) about
  - How to compile the program and run it
  - If there are some ideas interesting or skillful in your implementation , tell us that
  - Some other functions you implemented in your program but not on the specification. Also tell us how to use that
- Compress your report and source code as a file like b969XXXXX.rar (zip,7z,tar) and upload it to
  - FTP: 140.112.29.100
  - ID: cn2010
  - PW: ccfccf