2/23/2016

Assignment 3

Chat Application

Rounak Das
JADAVPUR UNIVERSITY, BCSE-IV (A2), ROLL-001210501036

Problem Statement:

Write multi-client chat application consisting of both client and server programs. In this chat application simultaneously several client can communicate with each other. For this you need a single server program that clients connect to. The client programs send the chat text (input) to the server and then the server distributes that chat text to all the other clients. Each client then displays the text sent to it by the server. Server should be able to handle several clients concurrently. It should work fine as clients come and go. This can be implemented in two ways

1. Server can handle multiple clients simultaneously by forking a separate process for each client.
2. Server can create separate threads (instead of processes) to handle separate clients.

In general, the server program:

- Accepts connection requests from clients
- For each accepted connection start a process/thread
- Each process/thread reads data from the client and sends it to all other clients or selected clients
- When a process/thread detects that a client has disconnected it should free allotted resources and stop processing for that client.

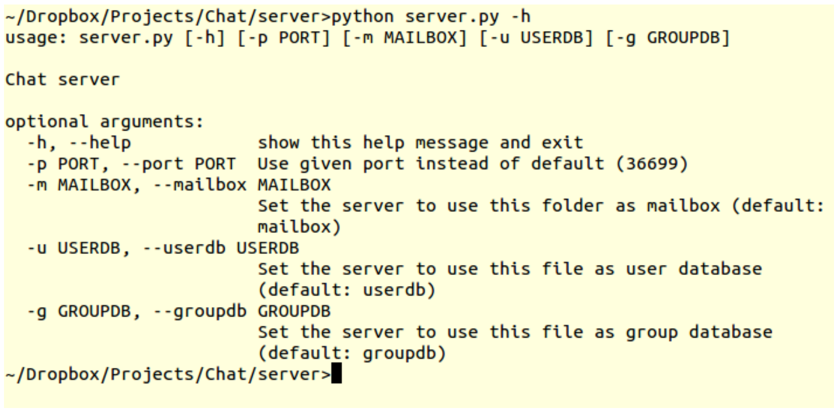
Design of the chat system:

The chat system is a centralized client-server application. The clients send their messaged to the server, which then relays the message to the client.

Server:

At the server the system mimics POP is some ways. The relay mechanism is based on files. Each unique user has their own file (hereafter referred to as their mailbox). The threads/processes deposit messages intended for a user in their mailbox. Access to this mailboxes are serialized using mutex locks. This is to protect mailboxes from concurrent access by threads/processes. When a user wants to read their messages their mailbox is read and the messages are sent to them. Thereafter the mailbox is emptied. The mailbox acts as a queue that stores one message per line.

Full server options:



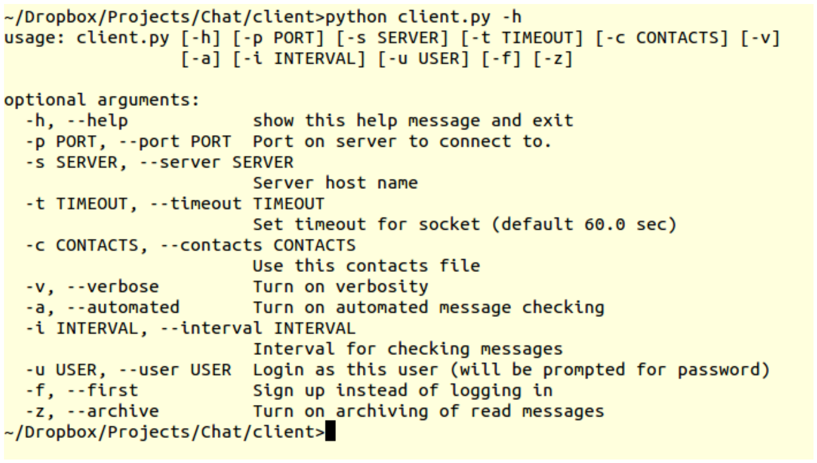
```
~/Dropbox/Projects/Chat/server>python server.py -h
usage: server.py [-h] [-p PORT] [-m MAILBOX] [-u USERDB] [-g GROUPDB]

Chat server

optional arguments:
  -h, --help            show this help message and exit
  -p PORT, --port PORT  Use given port instead of default (36699)
  -m MAILBOX, --mailbox MAILBOX
                        Set the server to use this folder as mailbox (default:
                        mailbox)
  -u USERDB, --userdb USERDB
                        Set the server to use this file as user database
                        (default: userdb)
  -g GROUPDB, --groupdb GROUPDB
                        Set the server to use this file as group database
                        (default: groupdb)
~/Dropbox/Projects/Chat/server>
```

- *userdb* is the file containing the list of users
- *groupdb* is the file containing all multicast group definitions
- *mailbox* is the folder where all the users' mailboxes are stored
- *port* can be reconfigured(default: 36699)

Client:

At the user side the client maintains a constant TCP connection with the server. It reads commands from the CLI and builds a message to the server by parsing the command, then sends it to the server. The server response is then showed to the user. Since messages that have been retrieved are deleted at the server, the client has an option to archive the messages that it reads. The client can be configured to poll the server at regular intervals to check if the user has any messages to read. By default the server IP and port are read from a configuration file "chat.cfg" but this can be overridden via command lien options. The client may also be run to sign up for this service.

Full client options:

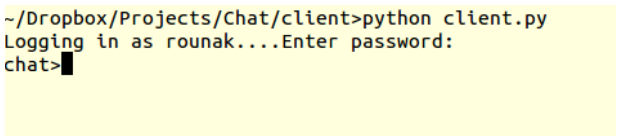


```
~/Dropbox/Projects/Chat/client>python client.py -h
usage: client.py [-h] [-p PORT] [-s SERVER] [-t TIMEOUT] [-c CONTACTS] [-v]
                 [-a] [-i INTERVAL] [-u USER] [-f] [-z]

optional arguments:
  -h, --help            show this help message and exit
  -p PORT, --port PORT  Port on server to connect to.
  -s SERVER, --server SERVER
                        Server host name
  -t TIMEOUT, --timeout TIMEOUT
                        Set timeout for socket (default 60.0 sec)
  -c CONTACTS, --contacts CONTACTS
                        Use this contacts file
  -v, --verbose         Turn on verbosity
  -a, --automated       Turn on automated message checking
  -i INTERVAL, --interval INTERVAL
                        Interval for checking messages
  -u USER, --user USER  Login as this user (will be prompted for password)
  -f, --first           Sign up instead of logging in
  -z, --archive         Turn on archiving of read messages
~/Dropbox/Projects/Chat/client>
```

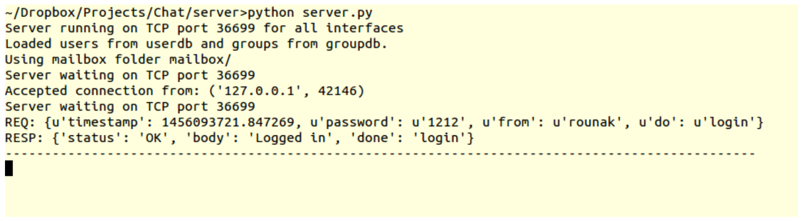
- server IP and port may be overridden here
- Timeout may be changed
- Contacts specifies a file from where local bindings of usernames to keywords can be saved; this is useful to remember who-is-who
- Verbose mode is for debugging
- If the client is set up to check for messages automatically, its polling interval can be changed
- If user is not specified in either "chat.cfg" or at the command line, the client prompts the user to specify a username
- If a new user wishes to login, they can specify the –f flag to sign up first
- Users may also all messages read during a session

All interaction between the server and the client happens via exchange of JSON strings encoded as base64 strings. Python dictionaries are used to build and store requests and responses. These are serialized to JSON strings before being encoded as base64 strings for sending.

The login interaction is shown below:

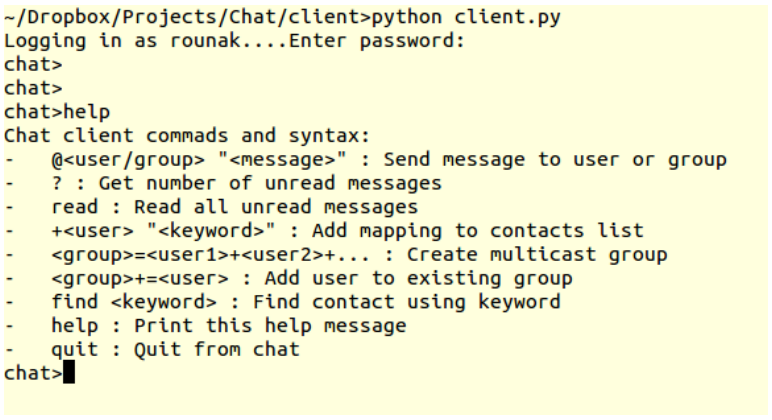


```
~/Dropbox/Projects/Chat/client>python client.py
Logging in as rounak....Enter password:
chat>
```



```
~/Dropbox/Projects/Chat/server>python server.py
Server running on TCP port 36699 for all interfaces
Loaded users from userdb and groups from groupdb.
Using mailbox folder mailbox/
Server waiting on TCP port 36699
Accepted connection from: ('127.0.0.1', 42146)
Server waiting on TCP port 36699
REQ: {u'timestamp': 1456093721.847269, u'password': u'1212', u'from': u'rounak', u'do': u'login'}
RESP: {'status': 'OK', 'body': 'Logged in', 'done': 'login'}
------------------------------------------------------------------------------------------------
```

The client provides an intuitive CLI for sending and receiving messages.

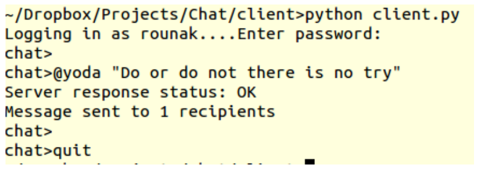


```
~/Dropbox/Projects/Chat/client>python client.py
Logging in as rounak....Enter password:
chat>
chat>
chat>help
Chat client commads and syntax:
-    @<user/group> "<message>" : Send message to user or group
-    ? : Get number of unread messages
-    read : Read all unread messages
-    +<user> "<keyword>" : Add mapping to contacts list
-    <group>=<user1>+<user2>+... : Create multicast group
-    <group>+=<user> : Add user to existing group
-    find <keyword> : Find contact using keyword
-    help : Print this help message
-    quit : Quit from chat
chat>
```

- *@user message*: sends a message to a user or group
- *?*: Gets the number of messages for the user without reading them
- *read*: reads the messages from the server
- *+user Keyword*: Add a local mapping of keyword to user
- *group=user1+user2+user3*: Creates a multicast group
- *group+=user*: Add a user to an existing group(only if you're already a member)
- *find Keyword*: Find local mapping for keyword
- *quit*: Exit

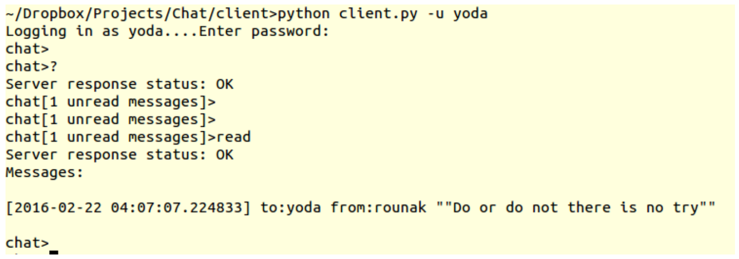
Example:

1. Send:



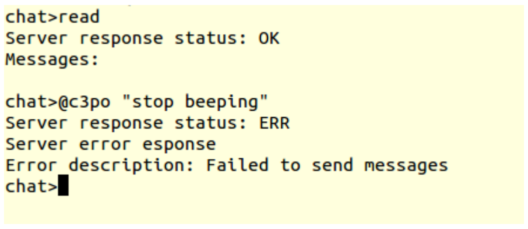
```
~/Dropbox/Projects/Chat/client>python client.py
Logging in as rounak....Enter password:
chat>
chat>@yoda "Do or do not there is no try"
Server response status: OK
Message sent to 1 recipients
chat>
chat>quit
```

2. Receive:



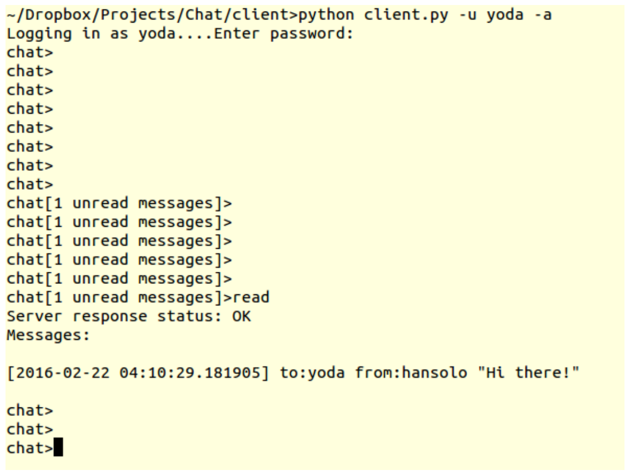
```
~/Dropbox/Projects/Chat/client>python client.py -u yoda
Logging in as yoda....Enter password:
chat>
chat>?
Server response status: OK
chat[1 unread messages]>
chat[1 unread messages]>
chat[1 unread messages]>read
Server response status: OK
Messages:

[2016-02-22 04:07:07.224833] to:yoda from:rounak ""Do or do not there is no try""

chat>_
```

3. OK and ERROR responses from server:



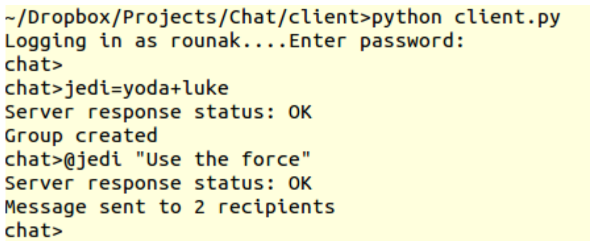
```
chat>read
Server response status: OK
Messages:

chat>@c3po "stop beeping"
Server response status: ERR
Server error esponse
Error description: Failed to send messages
chat>
```

4. Automated message checking:

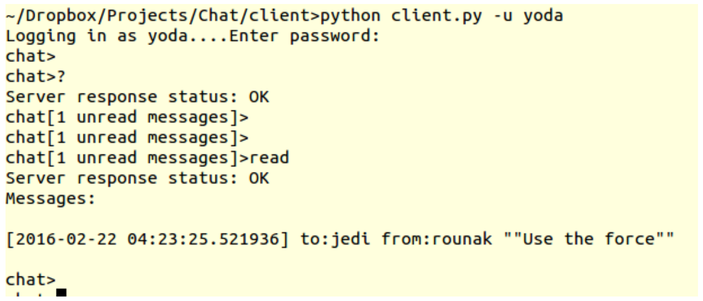


```
~/Dropbox/Projects/Chat/client>python client.py -u yoda -a
Logging in as yoda....Enter password:
chat>
chat>
chat>
chat>
chat>
chat>
chat>
chat>
chat[1 unread messages]>
chat[1 unread messages]>
chat[1 unread messages]>
chat[1 unread messages]>
chat[1 unread messages]>
chat[1 unread messages]>read
Server response status: OK
Messages:

[2016-02-22 04:10:29.181905] to:yoda from:hansolo "Hi there!"

chat>
chat>
chat>
```

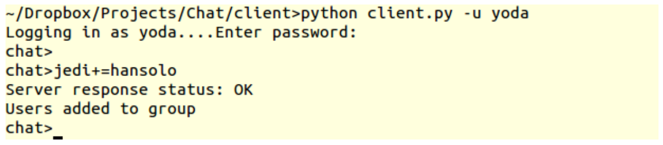
5. Group dynamics:



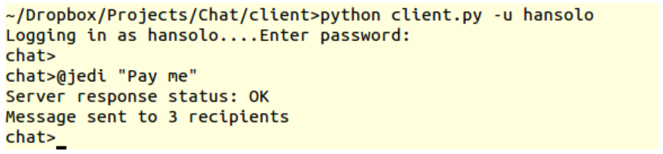
```
~/Dropbox/Projects/Chat/client>python client.py
Logging in as rounak....Enter password:
chat>
chat>jedi=yoda+luke
Server response status: OK
Group created
chat>@jedi "Use the force"
Server response status: OK
Message sent to 2 recipients
chat>
```



```
~/Dropbox/Projects/Chat/client>python client.py -u yoda
Logging in as yoda....Enter password:
chat>
chat>?
Server response status: OK
chat[1 unread messages]>
chat[1 unread messages]>
chat[1 unread messages]>read
Server response status: OK
Messages:

[2016-02-22 04:23:25.521936] to:jedi from:rounak ""Use the force""

chat>_
```

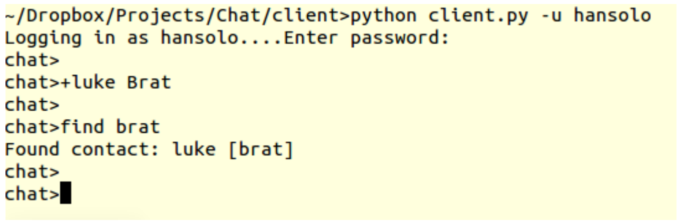


```
~/Dropbox/Projects/Chat/client>python client.py -u yoda
Logging in as yoda....Enter password:
chat>
chat>jedi+=hansolo
Server response status: OK
Users added to group
chat>_
```



```
~/Dropbox/Projects/Chat/client>python client.py -u hansolo
Logging in as hansolo....Enter password:
chat>
chat>@jedi "Pay me"
Server response status: OK
Message sent to 3 recipients
chat>_
```

6. Contacts:



```
~/Dropbox/Projects/Chat/client>python client.py -u hansolo
Logging in as hansolo....Enter password:
chat>
chat>+luke Brat
chat>
chat>find brat
Found contact: luke [brat]
chat>
chat>
```

Performance:

Settings:

Threads:

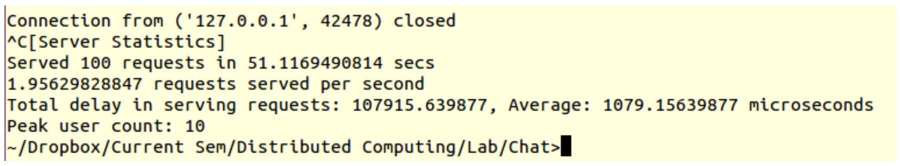
- Using pythons built-in `threading` module
- Using `threading.Lock` for mutex
- Each file has an associated lock
- Also locks for shared data structures

Processes:

- Using pythons `multiprocessing` module
- Using `multiprocessing`'s `Manager` based `Lock` object for synchronization
- The mailbox folder has one global lock
- Locks for shared variable and file handles

Testing methodology:

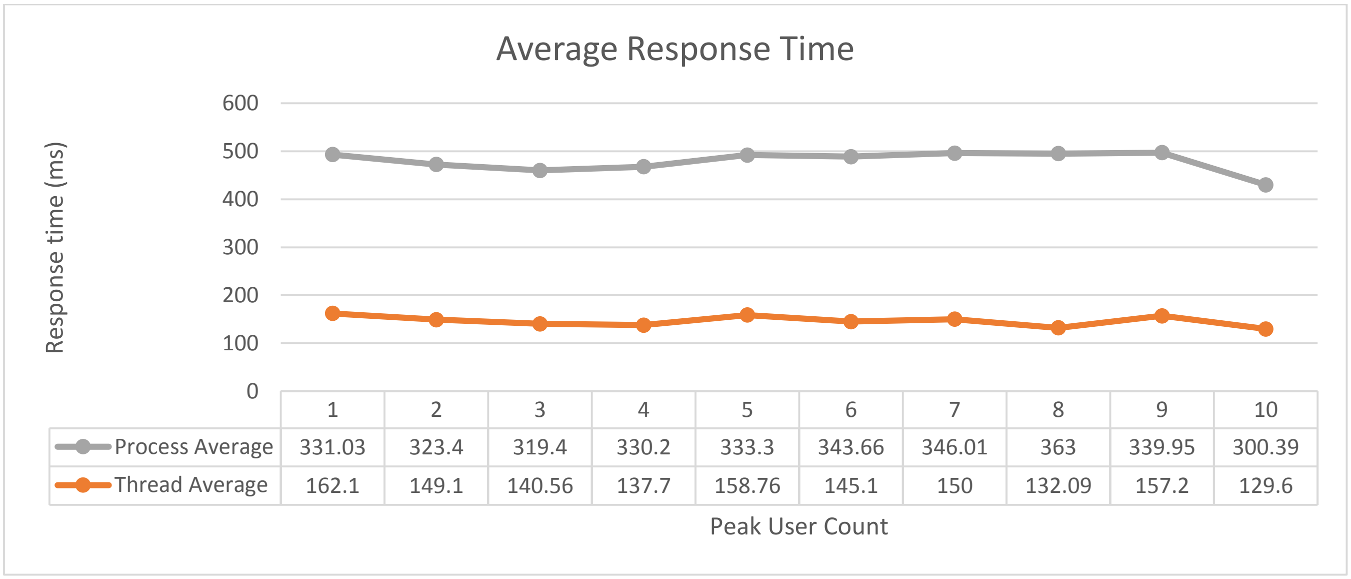
- Spawn separate processes one for each client
- Each client reads a set of commands from a file
- They send those commands to the server in quick succession (much quicker than humanly possible)
- At the server we measure:
  - Peak user count (number of concurrent threads/processes running at any time)
  - Total response time
  - Number of requests served
  - Uptime
  - E.g.



```
Connection from ('127.0.0.1', 42478) closed
^C[Server Statistics]
Served 100 requests in 51.1169490814 secs
1.95629828847 requests served per second
Total delay in serving requests: 107915.639877, Average: 1079.15639877 microseconds
Peak user count: 10
~/Dropbox/Current Sem/Distributed Computing/Lab/Chat>
```

Results:

The number of concurrent users was steadily decreased from 10 to 1 and we charted the average response time for each case thread vs process.



	1	2	3	4	5	6	7	8	9	10
Process Average	331.03	323.4	319.4	330.2	333.3	343.66	346.01	363	339.95	300.39
Thread Average	162.1	149.1	140.56	137.7	158.76	145.1	150	132.09	157.2	129.6

Conclusions:

The result shows the process takes about twice as long on average to respond than processes. This is due to a number of reasons:

- Inherent slowness of pythons Manager server process against threads
- Accessing shared memory objects take longer than data in your own address space
- Locks in shared memory are slower
- Processes are much higher overhead for something as simple as writing to files(which is what the chat server mostly does)
- Threads are faster for light computing jobs (such as this)
- Processes would be faster if our workload was more computing than IO. Separate processes would be scheduled so that they get more time on the CPU and increase throughput. For threads that must share CPU time among threads of the same process they will get lesser share of the CPU and will be slower.

Code:

Code is available at: https://github.com/DroidX86/CLIChat