P0P Design-Test Document

Partner A: Zain Bashir

Partner B: JB Ladera

1. Server Design

Overview

Describe the overall server design. You must include the number of threads you create including the main thread, what each thread is for and doing, any loops you have for each thread (and do this for both event-loop and thread-based server).

- Four threads
    - The main thread handles keyboard input.
    - handle_timer checks all sessions every five seconds in a while loop and removes those that have been inactive for more than five seconds.
    - handle_socket receives packets and modifies the state of the client according to the received packet. It also prints out the packet that was sent. After finishing the current packet, this thread loops back to the blocking receive packet call.
    - handle_print prints the queue of messages every two seconds.

Justification

Justify why your design is efficient and show evidence it can handle load efficiently. Specify the number of clients, the file size for each client, and the response rate/delay each client.

- Our design is efficient because for every packet received, the handle_socket thread will take the same amount of time. Extra work such as timers and printing is moved into separate threads where they will do any time intensive work.

```
c:\main\school\cs356\projects\p0_zain_and_jb (main -> origin)
λ node temp\analyze.js
Analyzing loss-cur-1.txt
data lines: 58941
Found 1 session id(s)
Individual loss rate: 0.726211483643274 (428 / 58936)
Overall loss rate: 0.726211483643274 (428 / 58936)

c:\main\school\cs356\projects\p0_zain_and_jb (main -> origin)
λ node temp\analyze.js
Analyzing loss-cur-2.txt
data lines: 117880
Found 2 session id(s)
Individual loss rate: 0.8348038550291842 (492 / 58936)
Individual loss rate: 0.8212298086059454 (484 / 58936)
Overall loss rate: 0.8280168318175648 (976 / 117872)

c:\main\school\cs356\projects\p0_zain_and_jb (main -> origin)
λ node temp\analyze.js
Analyzing loss-old-1.txt
data lines: 58941
Found 1 session id(s)
Individual loss rate: 0.5768969729876475 (340 / 58936)
Overall loss rate: 0.5768969729876475 (340 / 58936)

c:\main\school\cs356\projects\p0_zain_and_jb (main -> origin)
λ node temp\analyze.js
Analyzing loss-old-2.txt
data lines: 117880
Found 2 session id(s)
Individual loss rate: 2.6011266458531286 (1533 / 58936)
Individual loss rate: 2.6028234016560337 (1534 / 58936)
Overall loss rate: 2.6019750237545813 (3067 / 117872)
```

- In this image, loss-cur stands for our current version, and loss-old stands for our barebones version. Overall, for one client, the difference in loss rate was negligible. However, for two clients, our current version had less than ⅓ of the loss rate of our barebones version

Data structures

List any notable data structures you used and justify the use of the data structure. Also specify how you handle synchronization for the data structure if there were any need for synchronization.

- We have a queue to print from. This queue takes the messages in the order that they would have been sent to print, and handle_print prints them out. There is a lock on this queue so that it does not lose any messages when they are added to the queue from a different thread.
- We store all client sessions in a dictionary. This dictionary maps a session id to a ClientData class representing the state of the client. This data structure is

accessed from the handle_socket and handle_timer threads, so it needs to be synchronized.

How timeouts are handled

Describe how the timeouts are handled.

- Each client session has a field that keeps track of the last time a message was received from the client. If the handle_timer loop notices that this timer is longer than 5 seconds, then the session is closed.

How shutdown is handled

Describe how you handled the shutdown gracefully. (That is when you hit 'q' or ctrl+d in the command prompt.)

We shutdown gracefully by first setting a variable marking that the server is going to close. After this, we wait for the handle_timer and handle_print threads to finish which will close on their own once this variable is set. Then, all of the client sessions are closed and the print queue is emptied. handle_socket will either finish the current packet it is working on then block in receive packet, or it will finish the packet and break because it sees that the server is closing. If it is blocked and does not exit on its own, the socket is closed and the main thread exits, causing the thread to end since it is marked as a daemon. To handle a race condition with the daemon threads where a packet could be received while the server is closing, there is a lock preventing the packet from being processed if the server enters its closing state.

Any libraries used

List any libraries that you used for the implementation and justify the usage.

- N/A

Corner cases identified

Describe corner cases that you identified and how you handled them.

- We check the version and magic numbers for all incoming packets and discard the packet if they are incorrect
- Our print queue has a lock so that we do not lose messages through asynchrony.
- Sometimes, print is too fast for the terminal to handle and it crashes. We fix this through printing with a queue.
- Hello -> Data -> Hello vs Hello -> Hello -> Data. All three packets have the same sequence number. The first will close the session because it is an invalid state transition. The second will be ignored since it is a duplicate packet.
- Session closed -> GOODBYE from client should not be printed if the server receives a GOODBYE after the timer closes the session. This is handled by making sure that all of the prints are locked after receiving the packet.

2. Server Testing

Testing your server

Describe how you tested your server. Include the description of each test case and the setting (such as local only, one local one remote, how many clients, use of mock client, etc.).

3. Client Design

Overview

Describe the overall client design. Specify any differences between event-loop based client design and thread-based client design. You must include the number of threads you create including the main thread, what each thread is for and doing, any loops you have for each thread (and do this for both event-loop and thread-based client).

Our thread-based client has three threads: one that handles the socket, one that handles the keyboard, and one that handles the timer. The main thread blocks until one of these threads exits. Like our thread-based client, our event-based client has three separate events: the timer, keyboard, and socket. However, the socket thread gets remapped once the hello exchange is complete.

Justification

Justify why your design above is efficient. Also show evidence how your client can handle sending packets from a large file (each line is close to UDP packet max and also contains many lines) and receiving packets from the server at the same time. Note you do not want to cause the false TIMEOUT from server because you are too busy just sending out the packets to the server when the server actually has sent you a packet before the TIMEOUT.

Data structures

List any notable data structures you used and justify the use of the data structure. Also specify how you handle synchronization for the data structure if there were any need for synchronization.

We use a blocking queue that lets the main thread know that either the socket or keyboard want the program to exit. This is there so that the main thread exits the rest of the threads since they are marked as daemons.

How timeouts are handled

Describe how the timeouts are handled.

Timeouts are handled by an extra thread that sleeps for 5 seconds after a message is sent to the server. The timer is only set for the first message sent until a response is received.

How shutdown is handled

Describe how you handled the shutdown gracefully. (That is when you hit 'q' or ctrl+d in the command prompt.)

First, the close is called and the timer is restarted if a goodbye message is sent. This close function locks the rest of the threads. This will either cause the existing packet or text to finish being processed. Otherwise, the locks prevent any further packets or keyboard input to process. Then, once the closing thread has successfully locked the rest, it exits and fills the queue letting the main thread exit. Since these threads are not processing data, they are safe to exit. This locking is necessary since the threads must be marked as daemons due to their blocking calls. However, we do not want to exit the threads while they are processing data. Finally, the socket is closed once the client class is deconstructed.

Any libraries used

List any libraries that you used and justify the usage.

N/A

Corner cases identified

Describe corner cases that you identified and how you handled them.

Leaving any extra threads running significantly slows down python since python is single threaded where each thread runs on one core. This is why the main thread blocks instead of spinning in a while loop until a thread is no longer alive.

4. Client Testing

Testing your client

Describe how you tested your client. Include the description of each test case and the setting (such as local only, one local one remote, how many clients, use of mock server, etc.).

5. Reflection

Reflection on the process

What was most challenging in implementing the server? What was most challenging testing the server?

What was most challenging in implementing the client? What was most challenging testing the client?

What was most fun working on the project? What was most not-so-fun?

If you are to do this all over again how would you do it differently?

Reflection on pair programming

Log of the amount of time spent driving and the amount of time spent working individually for each part (e.g., X drives 1 hour; Y drives 45 minutes; X works alone for 1 hour, etc.)

- For Part 1
    - 2/12
        - Zain drives 45 minutes
        - JB drives 45 minutes
        - Zain drives 40 minutes
- For Part 2
    - 2/16
        - JB drives 45 minutes
        - Zain drives 45 minutes
        - JB drives 45 minutes
        - Zain drives 45 minutes
    - 2/17
        - JB drives 45 minutes
        - Zain drives 45 minutes
        - JB drives 40 minutes
    - 2/19
        - Zain drives 45 minutes
        - JB drives 45 minutes
- For Part 3
- For Part 4

What went well/or not-so-well doing pair programming? What was your take away in this process?

Submission

Remember to export to pdf and push it to your github team repo under the project root (the same level as README.txt)