Systems Programming – 2014/2015 - 2nd Semester

MMA - MEAer - MEEC - MEFT

Systems Programing — 2014/2015

Project

Students should implement a client-server based system that provides a simple chat service.

The clients connect to the server (using TCP IP sockets) and issue commands to identity the user, send chats and retrieve old chats.

The interaction between the client and server should be as follow:

- The client connects to the server using the regular socket functions
- The client sends the user identification
- The client sends chats (the text of those chats should be forwarded to all connected clients)
- The client disconnects
- During the connection the client can also request for the reception of older messages

The server should allow the concurrent connection of multiple client and should implement a minimal set of fault tolerance mechanisms.

During the project students should:

- define the architecture of the whole systems (client/server and server components)
- define the interfaces (exchanged messages) between client and server
- implement all defined components
- provide a set of tests to the chats storage component on the server.
- Evaluate the performance of the system

1 Client

The client should be a simple program that reads commands from the keyboard, sends requests and receives responses from the server

The commands received from the user should follow the structure:

- `LOGIN username` – log-ins into the server assigning a **username** to the connection
- `DISC` – disconnects and closes the connection with the server
- `CHAT string` – sends the **string** chat to the server
- `QUERY id_min id_max` – request old chats on the server with identifiers between **id_min** and **id_max**

A skeleton for the client will be provided.

The client should also present on the screen all chats sent by other clients.

2 Server

During its normal operation the server should:

- receive connection on port 3000 from the clients using sockets IP;
- receive messages from the connected clients and process them;
- read commands from the keyboard;
- implement a logging system.

2.1 Keyboard commands

The server can receive from the operator the following commands:

- `QUIT` – terminates orderly the server
- `LOG` – prints on the screen the log of all received messages

2.2 Message processing

When a message is receive, the server should decode, validate, process, send the result back to the client and store the relevant information on the log.

Each received messages should be numbered sequentially. This number should be stored in the log and sent in the response.

The identifiers for each messages is a positive integers. When the server starts, (nromally of from crash recovery) the first message will have the identifier of 1.

2.3 Chat storage

All chats sent by clients should be stored in memory storage (non persistent).

The chat storage does not need to be persistent: if the server crashes all chats are lost and an empty storage should be created.

The students should implement this component in a manner that it is possible to verify its functionality running an external tests suit.

This component should be implemented in a C module and provide a well defined interface that should be tested using UNITY tests.

2.4 Crash recover

When the server crashes the operation of the server should be resumed automatically without any operator intervention.

During crash recovery, it is note necessary to guarantee fault tolerance: during the recovery

period the server may be vulnerable to crashes.

2.5 Log system

The log component will store information regarding all received messages. This log should be persistent, and updated by new server instances. Besides the network messages receive the log should also contain information regarding the start, stop and recuperation of the server.

2.6 Server performance

Students should use the Linux parallel processing objects (threads and process) to implement concurrency and optimize the time necessary to produce and send responses to requests. For instance, the processing of a client message should not block the reception and processing of other messages, and the responses should be sent to the client as soon as possible, delaying to after the response sending the maximum of processing or relay some processing to concurrent tasks.

2.7 Modularity

Students should define the components of the server and try to implement them in a modular way. For instance all messages should be read/processed/sent by a set of functions that should for a coherent library (.c + .h files).

3 Messages

The students should use Protocol Buffer to define all messages exchanged between clients and server:

- LOGIN
- CHAT
- QUERY

3.1 LOGIN

The LOGIN message contains a proposed username.

As a response to the LOGIN message the server should inform the client if the proposes user is valid or not.

3.2 CHAT

The CHAT message should contains the chat text to forward to other clients.

The client that sends a CHAT should receive the same forwarded message as other clients.

3.3 QUERY

The QUERY message should contain two positive integers representing the limits of the query to be performed.

As response the server will send a message containing all chat text (with identifiers) between the sent ids.

3.4 Server messages

If additional messages are to be exchanged inside server components in order to implement concurrency and fault tolerance, they should also be defined using Protocol Buffers.

4 Project submission

The deadline for the project submission is Friday 22nd of May.

The submission will be performed using the FENIX system.

Students should submit the project code along with a report describing:

- System architecture
- Exchange messages
- Chat storage module test suit
- Performance enhancements
- Fault tolerance solutions

4.1 Project evaluation

The project will be evaluated taking into account the number of implemented functionalities along with the adopted solution:

Client – sending of messages

Client – reception of responses

Server – handling of operator keyboard option

Server – Crash recover

Server – Reception of messages

Server – Broadcast of chat text

Server – Storage of chat messages

Server – Response to QUERY message

Server – Log system

Server – Concurrency and performance enhancements

Chat storage – Unity test suit

System - Modularity

These points will be evaluated taking into account the produced code and its decsription of the report.