BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJASTHAN)

I SEMESTER 2024-2025

ASSIGNMENT
Course No.: IS F462 Course Title: Network Prog.

Group members: i) Jay Goyal: 2021A7PS2418P
ii) Yash Barge: 2021A7PS0006P

Problem statement P2.

Running the code

Compile the code by entering `gcc -o chat chat.c` . You may have to specify `-lrt` at the end to link to the POSIX realtime library.

Execute by running command `./chat` .

Chat server listens on port `12345`, hence you can connect to it using telnet by specifying the command `telnet 127.0.0.1 12345`.

Command list

Command list can be specified using the `help` command. An explanation of each command is:

`help`: Displays the help message.
`listu:` Displays all users, both offline and online.
`msgu <user-name> <message>`: Sends a message to the specified user.

`mkgrp <group-name> <user-list>`: Makes a group with the space separated list of users. You must specify your own username, else you will not be added to the group. **Group name cannot contain a space. Group must contain at least 3 members.**
`listg`: Lists all groups that the current user is a part of. **Does not display groups that the current user is not a part of.**
`msgg <group-name> <message>`: Sends a message to the specified group. **You cannot send messages to groups that you are not a part of.**
`msga <message>`: Sends a broadcast message to all users.

Design

(a) & (c)

`IDLE_CHILDREN` is specified using `#define`, and is listed near the top of the file. Can be changed to any value `>0`.

Parent process accepts connections on the listening socket, and distributes the file descriptor using UNIX domain sockets. All child processes listen on the same UNIX domain socket, and it is up to the scheduler to distribute them. Child processes will not accept more than `CLIENTS_PER_PROCESS`, which is also specified using `#define` near the top of the source file.

Child processes use event-driven processing for accepting data from client connections, sending data when it becomes available on the message queue (Linux only), and accepting new connections from the parent process over the UNIX domain socket. This is implemented using the `select()` call.

A shared mutex `socket_access_lock` regulates access to the UNIX domain socket over which connections are distributed. A child process acquires this mutex using the `pthread_mutex_trylock`. It can only acquire this mutex when it has fewer connections than `CLIENTS_PER_PROCESS`. As only one child process is woken up when a new connection is received, it avoids the thundering herd problem.

This program **only works when running on a Linux system**, as all the entire event loop is based on the `select()` call. The program uses POSIX message queues to distribute messages to users, which are not specified by POSIX to be file descriptors, but happen to be implemented that way on Linux. Hence, **it will only work on POSIX systems with realtime extensions which implement POSIX message queues using file descriptors.**

(b)

Using `sigaction` and `sigaction.sa_flags = SA_NOCLDWAIT`, when child processes exit they do not be `wait()`ed on by the parent. They do not become zombie processes.

Child processes monitor how many idle processes are there by updating the shared memory segment. If there are more than `IDLE_CHILDREN`, an idle child process will exit. If there are fewer than `IDLE_CHILDREN`, it will send a zero-byte message over the UNIX domain socket, informing the parent process to create new processes. The parent process monitors the UNIX domain socket and the TCP receiving socket using the events, implemented with the `select()` system call.

(d)

Usage of UNIX domain sockets has been described in the previous sections.

There is a shared memory segment that is used to access the list of users and multicast groups.

There is one POSIX message queue per user, that receives all the messages intended for that user. As it exists even when the user client is offline, messages can be sent to an offline user that will be shown when the user logs back on. Due to the finite size of message queues, when a lot of messages are pending for a user, there may not be any space left in the queue. As such, the earlier messages are discarded in favor of making the latest messages visible.

When the chat server is terminated using `Ctrl-C` (generating an interrupt), all POSIX message queues are unlinked and their contents destroyed.

(e)

When users join, they are asked to enter a username (spaces not allowed). If that username already exists, they are logged in as that user and are shown any messages they were sent while they were offline.

List of users with status is provided with `listu`.

Users can send messages to any other user with `msgu`, regardless of if they are online or not. If they are online, they will immediately receive the message. If they are not, they will receive it when they login.

Users can create groups with `mkgrp` and message groups with `msgg`, and send broadcast messages using `msga`. These messages will also be delivered to offline users, and will be displayed when they login.