P1. Chat Server

Design

When the server is run, it creates a TCP socket on the given port. It then listens on that port. Once that is done, it forks N times to create N child processes, and the child processes execute a process general “main” function. The parent then goes into an infinite loop that causes it to pause. The parent also registers a signal handler for the SIGUSR1 signal.

The process general “main” function keeps track of the number of threads it makes. It creates T threads initially. It uses a mutex variable to ensure that the condition it checks is checked while no other process/thread is modifying the variables involved. The threads it creates run its own thread general “main” function.

Each thread receives the socket file descriptor that the original process is listening on. It then accepts an incoming connection on that socket. It then waits for data on the connected socket. Upon receiving commands as input from the clients on the other end of the connection, the commands are parsed and the appropriate instructions are performed, allowing the clients to join, chat and leave. The name of the client and the port number that the client is connected on are stored in a temporary file during the execution of the program, so that the ‘chat’ command can work properly.

When the number of connections crosses the CPP given, the process sends a SIGUSR1 signal to the parent. The signal handler handles this by creating a new process to replace this one and runs the process general main function in the child.

The client connects to the given port and sends the user’s commands to the server. It creates a child process to receive incoming messages from the server.

Files Included

preft.c, client.c, Makefile

How to run

compile by running `make`
to start the chat server, `./preft <port> <N> <T> <CPP>`
to start the chat client, `./client <port>`

Ashwin Kiran Godbole
Samarth Krishna Murthy