Client Server Application

Ammar Lakho 18055

Help

Client

Input	**Result**
add/sub/mul/div <list>	Prints the answer on the screen
run <process>	Creates a new process and adds it to activeList and allList.
kill <pname>	Terminates the first instance of the process “pname”.
kill <pid>	Terminates the process with process ID=pid
listActive	Prints pid, name and start_time for each active process executed by the client.
listAll	Prints pid, name, start_time, end_time, and duration(in seconds) for each process executed by the client.

Server

Input	**Result**
listConn	Prints socketfd, IP and port# for each client.
print <msg>	Prints <msg> on each client’s terminal.
print <msg> <fd>	Prints <msg> on the terminal of the client with socketfd=fd.
listProcess	Prints the activeList for each client
listProcess <fd>	Prints the activeList for client with socketfd=fd.

How to Run

Client

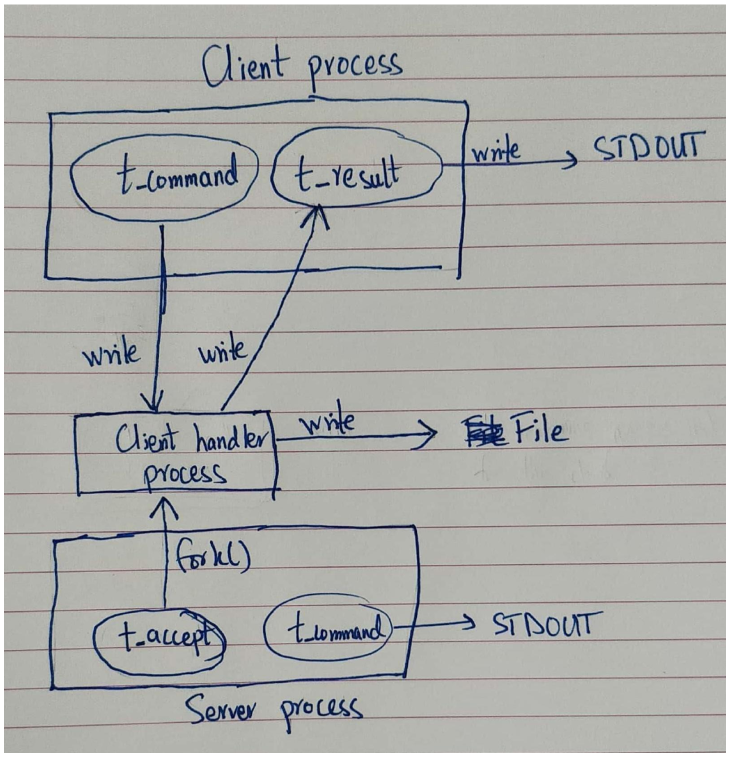
./client IP Port#

IP address of the computer running the server should be provided alongside the port that the server has made available for communication.

Server

./server Port#

Architecture



Key:
Oval = Thread
Rectangle = Process

Client

The client process connects to the server using an IP address and a port number provided as arguments.

After a successful connection, the client process breaks into 2 separate threads:

1. **Command Thread**: This thread reads a command from STDIN and writes it to the socket.
2. **Result Thread**: This thread reads the response from the server on the socket and writes it to STDOUT.

Server

The server process has 2 threads:

1. **Command Thread**: This thread reads a command from STDIN, understands the command, and writes a response to STDOUT.
2. **Accept Thread**: This thread accepts a connection from a client and if that is successful, it fork()s and the child process that is spawned becomes the client handler for the client that has just been accept()ed.

Client Handler: The client handler process reads from the socket to get the command entered by the client, understands it, and writes an appropriate response to the socket.

Achievements

1. Handled most errors and made the client aware of the error.
2. Handled unexpected termination of client process and client handler process.
3. If the main server process(conn) crashes, the connected clients still remain connected to the server(client handler) and can execute their commands. Only new connections won’t be handled.
4. Ensured no zombie processes exist to minimize wastage of resources.