**NETWORK ARCHITECHTURE-1**

**PROJECT**

**TEAM MEMBERS**

**NAME PROJECT ROLE**

SUNAYANA RAPOLU 2nd client

SAHITHI GADDAM 2nd server multi

LAVANYA PODA 1st question

**Part I.** **GENI/Socket programming Warm-up (50%)**

Develop and deploy a simple TCP client and server programs on GENI (refer GENI LabZero slide and sample socket programs). Show the screenshots of simple message exchanges. Start from client message ‘Hello from Client-your names’ and server responses with ‘Hello from Server-your names’. Then messages from each side are echoed to each other. The program quit the program with typing ‘Bye from Client-your name’ and ‘Bye from Server-your name’.

**CLIENT:**

/\*

C ECHO client example using sockets

Description:

This program will connect to localhost or 127.0.0.1 and port 8888. Then asks for commands to send.

Compile:

$ gcc EchoClient.c -o EchoClient

Here is an example, how the output would look

$./EchoClient

Socket created

Connected

Enter message : hi

Server reply :

hi

Enter message : how are you

\*/

#include<stdio.h> //printf

#include<string.h> //strlen

#include<sys/socket.h> //socket

#include<arpa/inet.h> //inet\_addr

int main(int argc , char \*argv[])

{

int sock;

struct sockaddr\_in server;

char message[10000] , server\_reply[20000];

char greetings1[1000] = "Hello from CLient - ";

char greetings2[1000] = "Bye from CLient - ";

char greetings3[1000] = "Bye from Server - ";

int loop = 1;

int firsttime = 1;

if (argc != 2)

{

printf("usage: %s <user name>\n", argv[0]);

return -1;

}

//Create socket

sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock == -1)

{

perror("Could not create socket\n");

}

printf("Socket created\n");

server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server.sin\_family = AF\_INET;

server.sin\_port = htons(8888);

//Connect to remote server

if (connect(sock , (struct sockaddr \*)&server , sizeof(server)) < 0)

{

perror("connect failed. Error");

return 1;

}

printf("Connected\n");

strncat(greetings1, argv[1], strlen(argv[1]));

//@printf("%s\n", greetings1);

strncat(greetings2, argv[1], strlen(argv[1]));

//@printf("%s\n", greetings2);

strncat(greetings3, argv[1], strlen(argv[1]));

//@printf("%s\n", greetings3);

do {

if(firsttime) {

strncpy(message, greetings1, strlen(greetings1));

firsttime = 0;

}

//Send initial data

if( send(sock, message, strlen(message), 0) < 0)

{

perror("Initial Send failed");

loop = 0;

break;

}

//Receive a reply from the server

if( recv(sock, server\_reply, 20000, 0) < 0)

{

perror("recv failed");

break;

}

printf("\nServer reply:");

printf(server\_reply);

memset(server\_reply, 0, sizeof(server\_reply));

if(strncmp(server\_reply, greetings3, strlen(greetings3)) == 0)

{

break;

}

memset(message, 0, sizeof(message));

printf("\nEnter message: ");

//scanf("%s", message);

fgets (message, sizeof(message), stdin);

} while (loop);

/\* //keep communicating with server

while(1)

{

memset(message, 0, sizeof(message));

printf("\nEnter message: ");

scanf("%s", message);

//Send some data

if( send(sock, message, strlen(message), 0) < 0)

{

perror("Send failed");

return 1;

}

//Receive a reply from the server

if( recv(sock, server\_reply, 20000, 0) < 0)

{

perror("recv failed");

break;

}

printf("\nServer reply:");

printf(server\_reply);

memset(server\_reply, 0, sizeof(server\_reply));

if(strncmp(server\_reply, greetings3, strlen(greetings3)) == 0)

{

break;

}

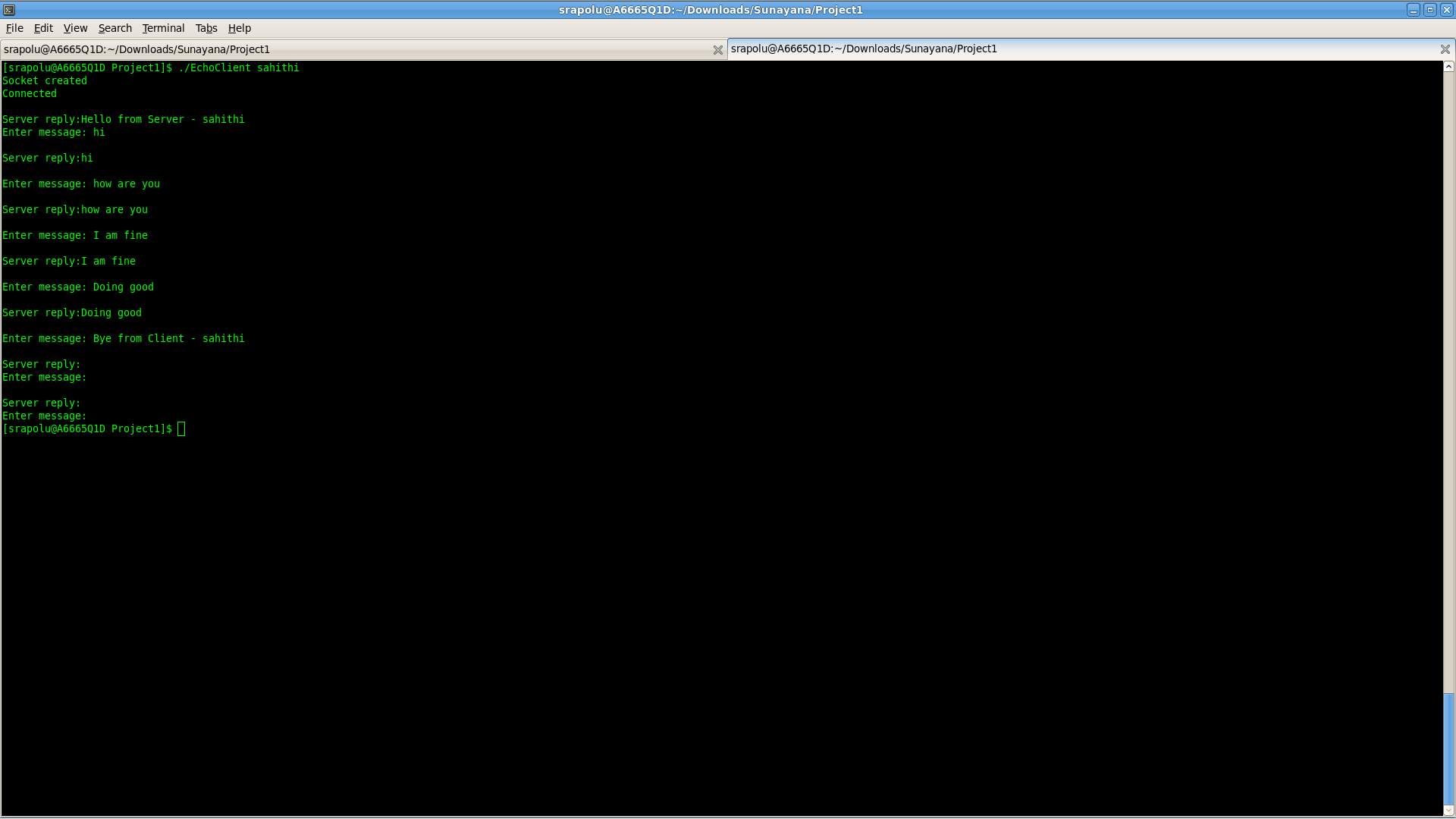
}

\*/

close(sock);

return 0;

}



**SERVER:**

/\*

C socket server example

Description:

This code will start a server on localhost or 127.0.0.1 and port 8888.

Once it receives a connection, it will read some input from the client and reply back with the same message.

Note: This server can handle communication with only 1 client at a time.

Compile:

$ gcc EchoServer.c -o EchoServer

To test the server, run the server and then connect from another terminal by using the "telnet" command like this:

$./EchoServer

$ telnet localhost 8888

\*/

#include<stdio.h>

#include<string.h> //strlen

#include<sys/socket.h>

#include<arpa/inet.h> //inet\_addr

#include<unistd.h> //write

int main(int argc, char \*argv[])

{

int socket\_desc, client\_sock, c, read\_size;

struct sockaddr\_in server, client;

char client\_message[20000];

char greetings1[1000] = "Hello from CLient - ";

char greetings2[1000] = "Hello from Server - ";

char greetings3[1000] = "Bye from Client - ";

if (argc != 2)

{

printf("usage: %s <user name>\n", argv[0]);

return -1;

}

//Create socket

socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0);

if (socket\_desc == -1)

{

perror("Could not create socket");

}

printf("Socket created\n");

//Prepare the sockaddr\_in structure

server.sin\_family = AF\_INET;

server.sin\_addr.s\_addr = INADDR\_ANY;

server.sin\_port = htons( 8888 );

//Bind

if( bind(socket\_desc, (struct sockaddr \*)&server, sizeof(server)) < 0)

{

//print the error message

perror("bind failed. Error");

return 1;

}

printf("bind done\n");

//Listen

listen(socket\_desc, 3);

//Accept and incoming connection

printf("Waiting for incoming connections...\n");

c = sizeof(struct sockaddr\_in);

//accept connection from an incoming client

client\_sock = accept(socket\_desc, (struct sockaddr \*)&client, (socklen\_t\*)&c);

if (client\_sock < 0)

{

perror("accept failed");

return 1;

}

printf("Connection accepted\n");

strncat(greetings1, argv[1], strlen(argv[1]));

//@printf("%s\n", greetings1);

strncat(greetings2, argv[1], strlen(argv[1]));

//@printf("%s\n", greetings2);

strncat(greetings3, argv[1], strlen(argv[1]));

//@printf("%s\n", greetings3);

//Receive a message from client

while( (read\_size = recv(client\_sock, client\_message, 20000, 0)) > 0 )

{

if (strncmp(client\_message, greetings1, strlen(greetings1)) == 0)

{

printf("Received %s from client and sending %s\n", client\_message, greetings2);

strncpy(client\_message, greetings2, strlen(greetings2));

}

else if (strncmp(client\_message, greetings3, strlen(greetings3)) == 0)

{

printf("Received %s from client and closing server\n", client\_message);

break;

}

//Send the message back to client

write(client\_sock, client\_message, strlen(client\_message));

printf("sending %s to client\n", client\_message);

memset(client\_message, 0, sizeof(client\_message));

}

if(read\_size == 0)

{

printf("Client disconnected\n");

fflush(stdout);

}

else if(read\_size == -1)

{

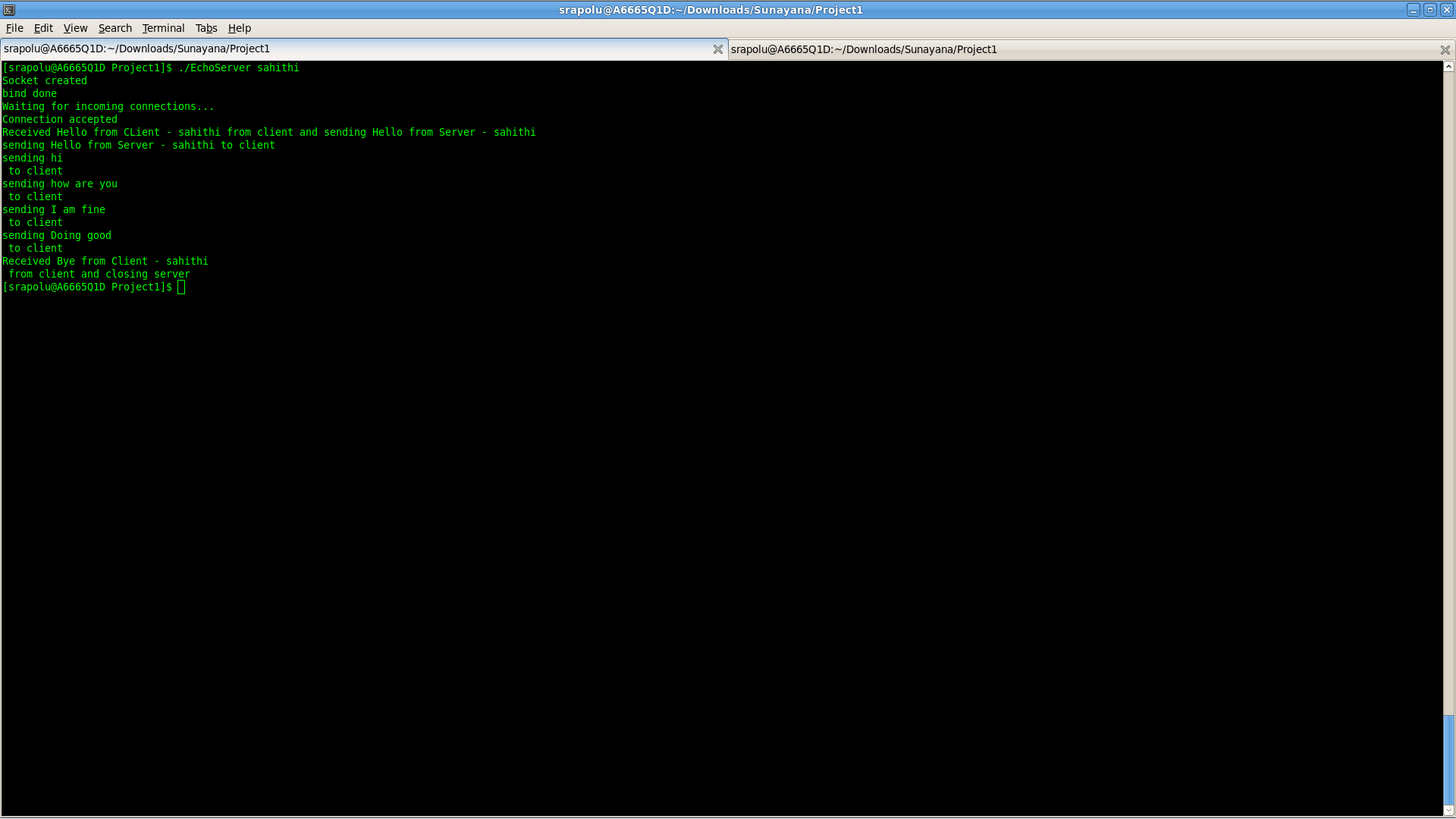
perror("recv failed");

}

close (client\_sock);

return 0;

}

****

**Part II. Simple Chat socket program (50%)**

Develop a simple chat program (similar to google hangout and skype chat), and show the screenshots of the execution of the below. Extend the first program to chat client-server program following these steps.

1. A chat server will accept a single client connection and display everything the client types. If the client user types ‘quit’, both client and server will end the program.

**CLIENT:**

/\*

C ECHO client example using sockets

Description:

This program will connect to localhost or 127.0.0.1 and port 8888. Then asks for commands to send.

Compile:

$ gcc EchoClient.c -o EchoClient

Here is an example, how the output would look

$./EchoClient

Socket created

Connected

Enter message : hi

Server reply :

hi

Enter message : how are you

\*/

#include<stdio.h> //printf

#include<string.h> //strlen

#include<sys/socket.h> //socket

#include<arpa/inet.h> //inet\_addr

int main(int argc , char \*argv[])

{

int sock;

struct sockaddr\_in server;

char message[10000] , server\_reply[20000];

int client\_id = 0;

char quit\_command[] = "quit";

if (argc != 2)

{

printf("%s <client ID>", argv[0]);

return -1;

}

client\_id = atoi(argv[1]);

printf("Received client id: %d\n", client\_id);

//Create socket

sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock == -1)

{

perror("Could not create socket");

}

printf("Socket created\n");

server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server.sin\_family = AF\_INET;

server.sin\_port = htons(8888);

//Connect to remote server

if (connect(sock , (struct sockaddr \*)&server , sizeof(server)) < 0)

{

perror("connect failed. Error");

return 1;

}

printf("Connected\n");

//keep communicating with server

while(1)

{

memset(message, 0, sizeof(message));

printf("Enter message for client ID#%d: ", client\_id);

fgets(message, sizeof(message), stdin);

//Send some data

if( send(sock, message, strlen(message), 0) < 0)

{

perror("Send failed");

break;;

}

if (strncmp(message, quit\_command, strlen(quit\_command)) == 0)

{

break;

}

//Receive a reply from the server

if( recv(sock, server\_reply, 20000, 0) < 0)

{

perror("recv failed");

break;

}

printf("Server reply for client ID#%d: ", client\_id);

printf(server\_reply);

memset(server\_reply, 0, sizeof(server\_reply));

}

close(sock);

return 0;

}

**SERVER MULTI:**

/\*

C socket server example, handles multiple clients using threads

Description:

This code will start a server on localhost or 127.0.0.1 and port 8888.

Once it receives a connection, it will read some input from the client and reply back with the same message.

To test the server, run the server and then connect from another terminal by using the "telnet" command like this:

$ gcc -o EchoServerMulti EchoServerMulti.c -lpthread

$./EchoServerMulti

$ telnet localhost 8888

\*/

#include<stdio.h>

#include<string.h> //strlen

#include<stdlib.h> //strlen

#include<sys/socket.h>

#include<arpa/inet.h> //inet\_addr

#include<unistd.h> //write

#include<pthread.h> //for threading , link with lpthread

//the thread function

void \*connection\_handler(void \*);

int main(int argc , char \*argv[])

{

int socket\_desc, client\_sock, c, \*new\_sock;

struct sockaddr\_in server, client;

//Create socket

socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0);

if (socket\_desc == -1)

{

perror("Could not create socket");

}

printf("Socket created\n");

//Prepare the sockaddr\_in structure

server.sin\_family = AF\_INET;

server.sin\_addr.s\_addr = INADDR\_ANY;

server.sin\_port = htons(8888);

//Bind

if( bind(socket\_desc, (struct sockaddr \*)&server, sizeof(server)) < 0)

{

//print the error message

perror("bind failed. Error");

return 1;

}

printf("bind done\n");

//Listen

listen(socket\_desc, 3);

//Accept and incoming connection

//@printf("Waiting for incoming connections...\n");

c = sizeof(struct sockaddr\_in);

//Accept and incoming connection

printf("Waiting for incoming connections...\n");

c = sizeof(struct sockaddr\_in);

while( (client\_sock = accept(socket\_desc, (struct sockaddr \*)&client, (socklen\_t\*)&c)) )

{

printf("Connection accepted\n");

pthread\_t sniffer\_thread;

new\_sock = malloc(1);

\*new\_sock = client\_sock;

if( pthread\_create( &sniffer\_thread, NULL, connection\_handler, (void\*) new\_sock) < 0)

{

perror("could not create thread");

return 1;

}

//Now join the thread , so that we dont terminate before the thread

//pthread\_join( sniffer\_thread , NULL);

printf("Handler assigned\n");

}

if (client\_sock < 0)

{

perror("accept failed");

return 1;

}

return 0;

}

/\*

\* This will handle connection for each client

\* \*/

void \*connection\_handler(void \*socket\_desc)

{

//Get the socket descriptor

int sock = \*(int\*)socket\_desc;

int read\_size;

char \*message, client\_message[20000];

char quit\_command[] = "quit";

//Send some messages to the client

/\*message = "Greetings! I am your connection handler\n";

write(sock, message, strlen(message));

message = "Now type something and I shall repeat what you type \n";

write(sock, message, strlen(message));\*/

//Receive a message from client

while( (read\_size = recv(sock, client\_message, 20000, 0)) > 0 )

{

//Send the message back to client

printf("Thread#%d=>Received message from client: %s\n", pthread\_self(), client\_message);

if (strncmp(client\_message, quit\_command, strlen(quit\_command)) == 0)

{

break;

}

write(sock, client\_message, strlen(client\_message));

memset(client\_message, 0, sizeof(client\_message));

}

if(read\_size == 0)

{

printf("Client disconnected\n");

fflush(stdout);

}

else if(read\_size == -1)

{

perror("recv failed");

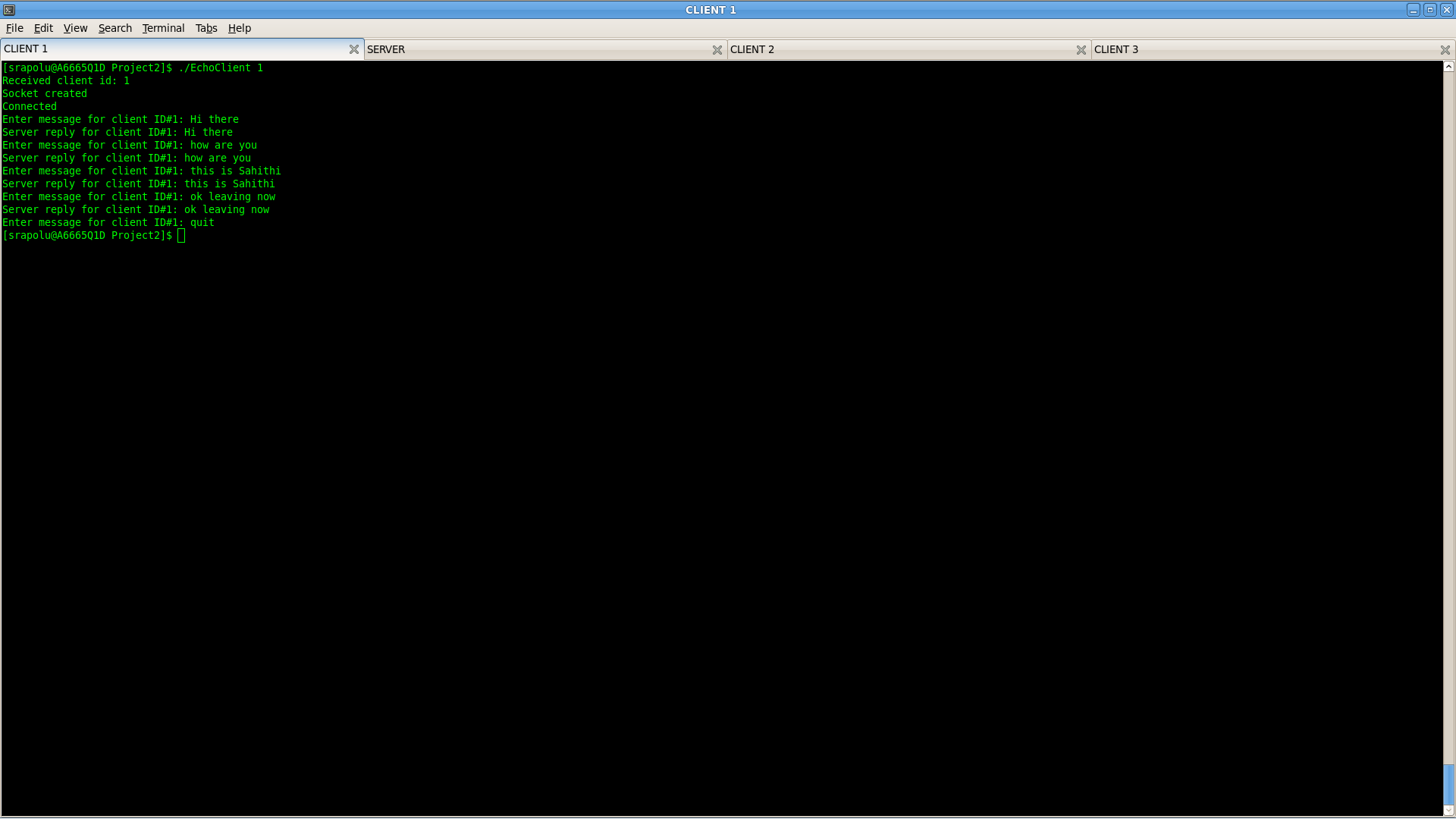
}

//Free the socket pointer

free(socket\_desc);

return 0;

}



1. A server now remains ‘open’ for additional connection once a client quits. The server can handle at most one connection at a time

**CLIENT:**

/\*

C ECHO client example using sockets

Description:

This program will connect to localhost or 127.0.0.1 and port 8888. Then asks for commands to send.

Compile:

$ gcc EchoClient.c -o EchoClient

Here is an example, how the output would look

$./EchoClient

Socket created

Connected

Enter message : hi

Server reply :

hi

Enter message : how are you

\*/

#include<stdio.h> //printf

#include<string.h> //strlen

#include<sys/socket.h> //socket

#include<arpa/inet.h> //inet\_addr

int main(int argc , char \*argv[])

{

int sock;

struct sockaddr\_in server;

char message[10000] , server\_reply[20000];

int client\_id = 0;

char quit\_command[] = "quit";

if (argc != 2)

{

printf("%s <client ID>", argv[0]);

return -1;

}

client\_id = atoi(argv[1]);

printf("Received client id: %d\n", client\_id);

//Create socket

sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock == -1)

{

perror("Could not create socket");

}

printf("Socket created\n");

server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server.sin\_family = AF\_INET;

server.sin\_port = htons(8888);

//Connect to remote server

if (connect(sock , (struct sockaddr \*)&server , sizeof(server)) < 0)

{

perror("connect failed. Error");

return 1;

}

printf("Connected\n");

//keep communicating with server

while(1)

{

memset(message, 0, sizeof(message));

printf("Enter message for client ID#%d: ", client\_id);

fgets(message, sizeof(message), stdin);

//Send some data

if( send(sock, message, strlen(message), 0) < 0)

{

perror("Send failed");

break;;

}

if (strncmp(message, quit\_command, strlen(quit\_command)) == 0)

{

break;

}

//Receive a reply from the server

if( recv(sock, server\_reply, 20000, 0) < 0)

{

perror("recv failed");

break;

}

printf("Server reply for client ID#%d: ", client\_id);

printf(server\_reply);

memset(server\_reply, 0, sizeof(server\_reply));

}

close(sock);

return 0;

}

**SERVER MULTI:**

/\*

C socket server example, handles multiple clients using threads

Description:

This code will start a server on localhost or 127.0.0.1 and port 8888.

Once it receives a connection, it will read some input from the client and reply back with the same message.

To test the server, run the server and then connect from another terminal by using the "telnet" command like this:

$ gcc -o EchoServerMulti EchoServerMulti.c -lpthread

$./EchoServerMulti

$ telnet localhost 8888

\*/

#include<stdio.h>

#include<string.h> //strlen

#include<stdlib.h> //strlen

#include<sys/socket.h>

#include<arpa/inet.h> //inet\_addr

#include<unistd.h> //write

#include<pthread.h> //for threading , link with lpthread

//the thread function

void \*connection\_handler(void \*);

int main(int argc , char \*argv[])

{

int socket\_desc, client\_sock, c, \*new\_sock;

struct sockaddr\_in server, client;

//Create socket

socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0);

if (socket\_desc == -1)

{

perror("Could not create socket");

}

printf("Socket created\n");

//Prepare the sockaddr\_in structure

server.sin\_family = AF\_INET;

server.sin\_addr.s\_addr = INADDR\_ANY;

server.sin\_port = htons(8888);

//Bind

if( bind(socket\_desc, (struct sockaddr \*)&server, sizeof(server)) < 0)

{

//print the error message

perror("bind failed. Error");

return 1;

}

printf("bind done\n");

//Listen

listen(socket\_desc, 3);

//Accept and incoming connection

//@printf("Waiting for incoming connections...\n");

c = sizeof(struct sockaddr\_in);

//Accept and incoming connection

printf("Waiting for incoming connections...\n");

c = sizeof(struct sockaddr\_in);

while( (client\_sock = accept(socket\_desc, (struct sockaddr \*)&client, (socklen\_t\*)&c)) )

{

printf("Connection accepted\n");

pthread\_t sniffer\_thread;

new\_sock = malloc(1);

\*new\_sock = client\_sock;

if( pthread\_create( &sniffer\_thread, NULL, connection\_handler, (void\*) new\_sock) < 0)

{

perror("could not create thread");

return 1;

}

//Now join the thread , so that we dont terminate before the thread

//pthread\_join( sniffer\_thread , NULL);

printf("Handler assigned\n");

}

if (client\_sock < 0)

{

perror("accept failed");

return 1;

}

return 0;

}

/\*

\* This will handle connection for each client

\* \*/

void \*connection\_handler(void \*socket\_desc)

{

//Get the socket descriptor

int sock = \*(int\*)socket\_desc;

int read\_size;

char \*message, client\_message[20000];

char quit\_command[] = "quit";

//Send some messages to the client

/\*message = "Greetings! I am your connection handler\n";

write(sock, message, strlen(message));

message = "Now type something and I shall repeat what you type \n";

write(sock, message, strlen(message));\*/

//Receive a message from client

while( (read\_size = recv(sock, client\_message, 20000, 0)) > 0 )

{

//Send the message back to client

printf("Thread#%d=>Received message from client: %s\n", pthread\_self(), client\_message);

if (strncmp(client\_message, quit\_command, strlen(quit\_command)) == 0)

{

break;

}

write(sock, client\_message, strlen(client\_message));

memset(client\_message, 0, sizeof(client\_message));

}

if(read\_size == 0)

{

printf("Client disconnected\n");

fflush(stdout);

}

else if(read\_size == -1)

{

perror("recv failed");

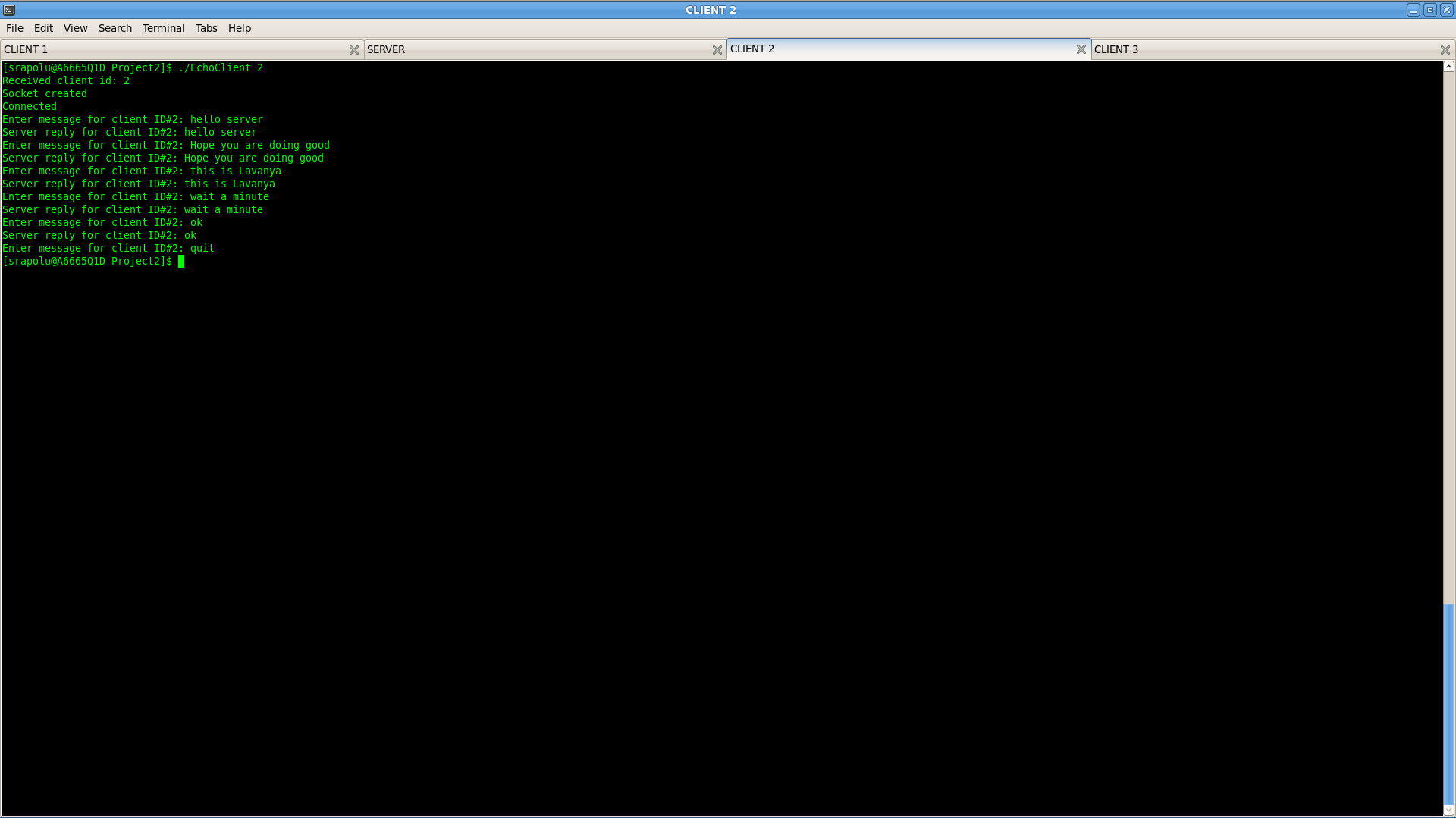
}

//Free the socket pointer

free(socket\_desc);

return 0;

}

**c)** A server now can handle multiple clients at the same time. The output from all the connected clients will appear on the server’s screen.

**CLIENT:**

/\*

C ECHO client example using sockets

Description:

This program will connect to localhost or 127.0.0.1 and port 8888. Then asks for commands to send.

Compile:

$ gcc EchoClient.c -o EchoClient

Here is an example, how the output would look

$./EchoClient

Socket created

Connected

Enter message : hi

Server reply :

hi

Enter message : how are you

\*/

#include<stdio.h> //printf

#include<string.h> //strlen

#include<sys/socket.h> //socket

#include<arpa/inet.h> //inet\_addr

int main(int argc , char \*argv[])

{

int sock;

struct sockaddr\_in server;

char message[10000] , server\_reply[20000];

int client\_id = 0;

char quit\_command[] = "quit";

if (argc != 2)

{

printf("%s <client ID>", argv[0]);

return -1;

}

client\_id = atoi(argv[1]);

printf("Received client id: %d\n", client\_id);

//Create socket

sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock == -1)

{

perror("Could not create socket");

}

printf("Socket created\n");

server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server.sin\_family = AF\_INET;

server.sin\_port = htons(8888);

//Connect to remote server

if (connect(sock , (struct sockaddr \*)&server , sizeof(server)) < 0)

{

perror("connect failed. Error");

return 1;

}

printf("Connected\n");

//keep communicating with server

while(1)

{

memset(message, 0, sizeof(message));

printf("Enter message for client ID#%d: ", client\_id);

fgets(message, sizeof(message), stdin);

//Send some data

if( send(sock, message, strlen(message), 0) < 0)

{

perror("Send failed");

break;;

}

if (strncmp(message, quit\_command, strlen(quit\_command)) == 0)

{

break;

}

//Receive a reply from the server

if( recv(sock, server\_reply, 20000, 0) < 0)

{

perror("recv failed");

break;

}

printf("Server reply for client ID#%d: ", client\_id);

printf(server\_reply);

memset(server\_reply, 0, sizeof(server\_reply));

}

close(sock);

return 0;

}

**SERVER MULTI:**

/\*

C socket server example, handles multiple clients using threads

Description:

This code will start a server on localhost or 127.0.0.1 and port 8888.

Once it receives a connection, it will read some input from the client and reply back with the same message.

To test the server, run the server and then connect from another terminal by using the "telnet" command like this:

$ gcc -o EchoServerMulti EchoServerMulti.c -lpthread

$./EchoServerMulti

$ telnet localhost 8888

\*/

#include<stdio.h>

#include<string.h> //strlen

#include<stdlib.h> //strlen

#include<sys/socket.h>

#include<arpa/inet.h> //inet\_addr

#include<unistd.h> //write

#include<pthread.h> //for threading , link with lpthread

//the thread function

void \*connection\_handler(void \*);

int main(int argc , char \*argv[])

{

int socket\_desc, client\_sock, c, \*new\_sock;

struct sockaddr\_in server, client;

//Create socket

socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0);

if (socket\_desc == -1)

{

perror("Could not create socket");

}

printf("Socket created\n");

//Prepare the sockaddr\_in structure

server.sin\_family = AF\_INET;

server.sin\_addr.s\_addr = INADDR\_ANY;

server.sin\_port = htons(8888);

//Bind

if( bind(socket\_desc, (struct sockaddr \*)&server, sizeof(server)) < 0)

{

//print the error message

perror("bind failed. Error");

return 1;

}

printf("bind done\n");

//Listen

listen(socket\_desc, 3);

//Accept and incoming connection

//@printf("Waiting for incoming connections...\n");

c = sizeof(struct sockaddr\_in);

//Accept and incoming connection

printf("Waiting for incoming connections...\n");

c = sizeof(struct sockaddr\_in);

while( (client\_sock = accept(socket\_desc, (struct sockaddr \*)&client, (socklen\_t\*)&c)) )

{

printf("Connection accepted\n");

pthread\_t sniffer\_thread;

new\_sock = malloc(1);

\*new\_sock = client\_sock;

if( pthread\_create( &sniffer\_thread, NULL, connection\_handler, (void\*) new\_sock) < 0)

{

perror("could not create thread");

return 1;

}

//Now join the thread , so that we dont terminate before the thread

//pthread\_join( sniffer\_thread , NULL);

printf("Handler assigned\n");

}

if (client\_sock < 0)

{

perror("accept failed");

return 1;

}

return 0;

}

/\*

\* This will handle connection for each client

\* \*/

void \*connection\_handler(void \*socket\_desc)

{

//Get the socket descriptor

int sock = \*(int\*)socket\_desc;

int read\_size;

char \*message, client\_message[20000];

char quit\_command[] = "quit";

//Send some messages to the client

/\*message = "Greetings! I am your connection handler\n";

write(sock, message, strlen(message));

message = "Now type something and I shall repeat what you type \n";

write(sock, message, strlen(message));\*/

//Receive a message from client

while( (read\_size = recv(sock, client\_message, 20000, 0)) > 0 )

{

//Send the message back to client

printf("Thread#%d=>Received message from client: %s\n", pthread\_self(), client\_message);

if (strncmp(client\_message, quit\_command, strlen(quit\_command)) == 0)

{

break;

}

write(sock, client\_message, strlen(client\_message));

memset(client\_message, 0, sizeof(client\_message));

}

if(read\_size == 0)

{

printf("Client disconnected\n");

fflush(stdout);

}

else if(read\_size == -1)

{

perror("recv failed");

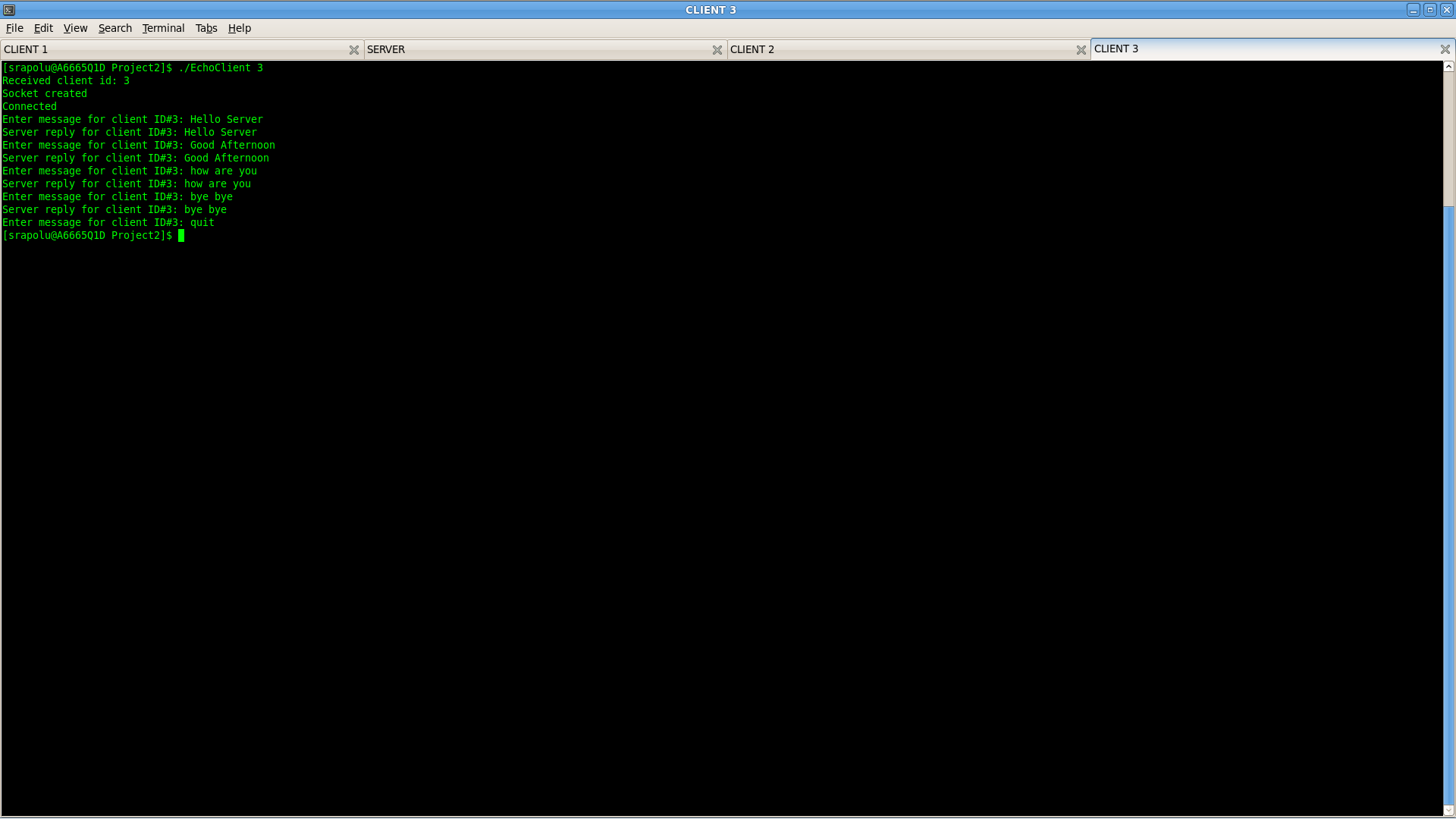
}

//Free the socket pointer

free(socket\_desc);

return 0;

}



**d)** A server now echoes all the text received from any of the connected clients to all.

**CLIENT:**

/\*

C ECHO client example using sockets

Description:

This program will connect to localhost or 127.0.0.1 and port 8888. Then asks for commands to send.

Compile:

$ gcc EchoClient.c -o EchoClient

Here is an example, how the output would look

$./EchoClient

Socket created

Connected

Enter message : hi

Server reply :

hi

Enter message : how are you

\*/

#include<stdio.h> //printf

#include<string.h> //strlen

#include<sys/socket.h> //socket

#include<arpa/inet.h> //inet\_addr

int main(int argc , char \*argv[])

{

int sock;

struct sockaddr\_in server;

char message[10000] , server\_reply[20000];

int client\_id = 0;

char quit\_command[] = "quit";

if (argc != 2)

{

printf("%s <client ID>", argv[0]);

return -1;

}

client\_id = atoi(argv[1]);

printf("Received client id: %d\n", client\_id);

//Create socket

sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock == -1)

{

perror("Could not create socket");

}

printf("Socket created\n");

server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server.sin\_family = AF\_INET;

server.sin\_port = htons(8888);

//Connect to remote server

if (connect(sock , (struct sockaddr \*)&server , sizeof(server)) < 0)

{

perror("connect failed. Error");

return 1;

}

printf("Connected\n");

//keep communicating with server

while(1)

{

memset(message, 0, sizeof(message));

printf("Enter message for client ID#%d: ", client\_id);

fgets(message, sizeof(message), stdin);

//Send some data

if( send(sock, message, strlen(message), 0) < 0)

{

perror("Send failed");

break;;

}

if (strncmp(message, quit\_command, strlen(quit\_command)) == 0)

{

break;

}

//Receive a reply from the server

if( recv(sock, server\_reply, 20000, 0) < 0)

{

perror("recv failed");

break;

}

printf("Server reply for client ID#%d: ", client\_id);

printf(server\_reply);

memset(server\_reply, 0, sizeof(server\_reply));

}

close(sock);

return 0;

}

**SERVER MULTI:**

/\*

C socket server example, handles multiple clients using threads

Description:

This code will start a server on localhost or 127.0.0.1 and port 8888.

Once it receives a connection, it will read some input from the client and reply back with the same message.

To test the server, run the server and then connect from another terminal by using the "telnet" command like this:

$ gcc -o EchoServerMulti EchoServerMulti.c -lpthread

$./EchoServerMulti

$ telnet localhost 8888

\*/

#include<stdio.h>

#include<string.h> //strlen

#include<stdlib.h> //strlen

#include<sys/socket.h>

#include<arpa/inet.h> //inet\_addr

#include<unistd.h> //write

#include<pthread.h> //for threading , link with lpthread

//the thread function

void \*connection\_handler(void \*);

int main(int argc , char \*argv[])

{

int socket\_desc, client\_sock, c, \*new\_sock;

struct sockaddr\_in server, client;

//Create socket

socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0);

if (socket\_desc == -1)

{

perror("Could not create socket");

}

printf("Socket created\n");

//Prepare the sockaddr\_in structure

server.sin\_family = AF\_INET;

server.sin\_addr.s\_addr = INADDR\_ANY;

server.sin\_port = htons(8888);

//Bind

if( bind(socket\_desc, (struct sockaddr \*)&server, sizeof(server)) < 0)

{

//print the error message

perror("bind failed. Error");

return 1;

}

printf("bind done\n");

//Listen

listen(socket\_desc, 3);

//Accept and incoming connection

//@printf("Waiting for incoming connections...\n");

c = sizeof(struct sockaddr\_in);

//Accept and incoming connection

printf("Waiting for incoming connections...\n");

c = sizeof(struct sockaddr\_in);

while( (client\_sock = accept(socket\_desc, (struct sockaddr \*)&client, (socklen\_t\*)&c)) )

{

printf("Connection accepted\n");

pthread\_t sniffer\_thread;

new\_sock = malloc(1);

\*new\_sock = client\_sock;

if( pthread\_create( &sniffer\_thread, NULL, connection\_handler, (void\*) new\_sock) < 0)

{

perror("could not create thread");

return 1;

}

//Now join the thread , so that we dont terminate before the thread

//pthread\_join( sniffer\_thread , NULL);

printf("Handler assigned\n");

}

if (client\_sock < 0)

{

perror("accept failed");

return 1;

}

return 0;

}

/\*

\* This will handle connection for each client

\* \*/

void \*connection\_handler(void \*socket\_desc)

{

//Get the socket descriptor

int sock = \*(int\*)socket\_desc;

int read\_size;

char \*message, client\_message[20000];

char quit\_command[] = "quit";

//Send some messages to the client

/\*message = "Greetings! I am your connection handler\n";

write(sock, message, strlen(message));

message = "Now type something and I shall repeat what you type \n";

write(sock, message, strlen(message));\*/

//Receive a message from client

while( (read\_size = recv(sock, client\_message, 20000, 0)) > 0 )

{

//Send the message back to client

printf("Thread#%d=>Received message from client: %s\n", pthread\_self(), client\_message);

if (strncmp(client\_message, quit\_command, strlen(quit\_command)) == 0)

{

break;

}

write(sock, client\_message, strlen(client\_message));

memset(client\_message, 0, sizeof(client\_message));

}

if(read\_size == 0)

{

printf("Client disconnected\n");

fflush(stdout);

}

else if(read\_size == -1)

{

perror("recv failed");

}

//Free the socket pointer

free(socket\_desc);

return 0;

}

