**9. IPC: Socket**

**9.3** Write two programs (server and client) to show how you canestablish a TCP socket connection using the above functions.

**OBJECTIVES:**

1. To learn about fundamentals of IPC through C socket programming.
2. Learn and understand the OS intraction with socket programming.
3. Use of system call and IPC mechanism to write effective application programs.
4. To know the port numbersing and process relation.
5. To knows the iterative and concurrent server concept.

**THEORY:**

JAVA SOCKET PROGRAMMING

A very basic one-way Client and Server setup where a Client connects, sends messages to server and the server shows them using socket connection. Java API networking package (java.net) takes care of all of that, making network programming very easy for programmers.

CLIENT SIDE PROGRAMMING:

Establish a Socket Connection

To connect to other machine we need a socket connection. A socket connection means the two machines have information about each other’s network location (IP Address) and TCP port. The java.net.Socket class represents a Socket. To open a socket:

Socket socket = new Socket(“127.0.0.1”, 5000)

* First argument – IP address of Server. (127.0.0.1 is the IP address of localhost, where code will run on single stand-alone machine).
* Second argument – TCP Port. (Just a number representing which application to run on a

server. For example, HTTP runs on port 80. Port number can be from 0 to 65535)

To communicate over a socket connection, streams are used to both input and output the data.

Closing the connection

The socket connection is closed explicitly once the message to server is sent.

SERVER SIDE PROGRAMMING:

Establish a Socket Connection

To write a server application two sockets are needed.

* A ServerSocket which waits for the client requests (when a client makes a new Socket())
* A plain old Socket socket to use for communication with the client.

getOutputStream() method is used to send the output through the socket.

Close the Connection

After finishing, it is important to close the connection by closing the socket as well as input/output streams.

**Program:**

**SERVER:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define SERV\_TCP\_PORT 8000 /\* server's port number \*/ #define MAX\_SIZE 80

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

{

int sockfd, newsockfd, clilen;

struct sockaddr\_in cli\_addr, serv\_addr;

int port;

char string[MAX\_SIZE];

int len;

/\* command line: server [port\_number] \*/

if(argc == 2)

sscanf(argv[1], "%d", &port); /\* read the port number if provided \*/ else

port = SERV\_TCP\_PORT;

/\* open a TCP socket (an Internet stream socket) \*/ if((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) { perror("can't open stream socket"); exit(1);

}

/\* bind the local address, so that the cliend can send to server \*/

bzero((char \*) &serv\_addr, sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

serv\_addr.sin\_port = htons(port);

if(bind(sockfd, (struct sockaddr \*) &serv\_addr, sizeof(serv\_addr)) < 0) { perror("can't bind local address");

exit(1);

}

/\* listen to the socket \*/

listen(sockfd, 5);

for(;;) {

/\* wait for a connection from a client; this is an iterative server \*/ clilen = sizeof(cli\_addr);

newsockfd = accept(sockfd, (struct sockaddr \*) &cli\_addr, &clilen); if(newsockfd < 0) {

perror("can't bind local address");

}

/\* read a message from the client \*/

len = read(newsockfd, string, MAX\_SIZE);

/\* make sure it's a proper string \*/

string[len] = 0;

printf("%s\n", string);

close(newsockfd);

}

}

**CLIENT:**

#include <string.h>

#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#include <netinet/tcp.h>

#include <netdb.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#define SERV\_TCP\_PORT 8000 /\* server's port \*/ int main(int argc, char \*argv[]) {

int sockfd;

struct sockaddr\_in serv\_addr;

char \*serv\_host = "localhost";

struct hostent \*host\_ptr;

int port;

int buff\_size = 0;

/\* command line: client [host [port]]\*/

if(argc >= 2)

serv\_host = argv[1]; /\* read the host if provided \*/ if(argc == 3)

sscanf(argv[2], "%d", &port); /\* read the port if provided \*/ else

port = SERV\_TCP\_PORT;

/\* get the address of the host \*/

if((host\_ptr = gethostbyname(serv\_host)) == NULL) { perror("gethostbyname error"); exit(1);

}

if(host\_ptr->h\_addrtype != AF\_INET) {

perror("unknown address type");

exit(1);

}

bzero((char \*) &serv\_addr, sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr =

((struct in\_addr \*)host\_ptr->h\_addr\_list[0])->s\_addr; serv\_addr.sin\_port = htons(port); /\* open a TCP socket \*/

if((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) { perror("can't open stream socket"); exit(1);

/\* connect to the server \*/

if(connect(sockfd, (struct sockaddr \*) &serv\_addr, sizeof(serv\_addr)) < 0) {

perror("can't connect to server");

exit(1);

}

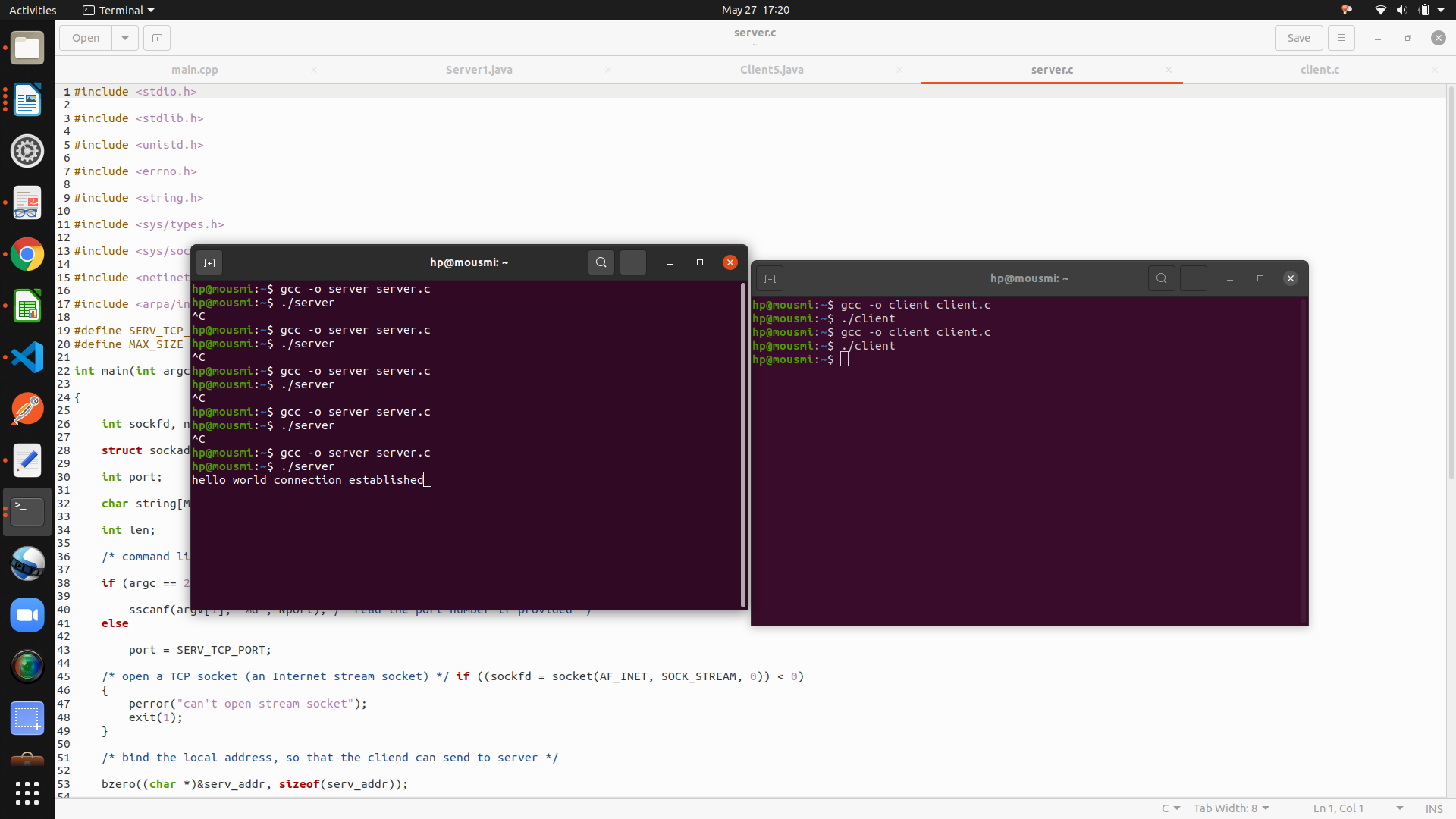
/\* write a message to the server \*/

write(sockfd, "hello world connetion established", sizeof("hello world connetion established"));

close(sockfd);

}

**OUTPUT:-**



**Conclusion:**

1.TCP socket connetion using system calls in C studied

2. Client server connection established.

**References:**

[www.cs.cf.ac.uk/Dave/C/CE.html](http://www.cs.cf.ac.uk/Dave/C/CE.html)