ECEN 602: Network Programming Assignment 1

TCP Echo Server and Client

TEAM DESCRIPTION:

TEAM NUMBER: 11

1) Srividhya Balaji UIN : 827007169
2) Sanjana Srinivasan UIN: 927008860

PROBLEM STATEMENT: To implement client and server for a simple TCP Echo service.

INDIVIDUAL TEAM MEMBER CONTRIBUTION:

Srividhya Balaji : Client implementation

Sanjana Srinivasan : Server implementation

FILES:

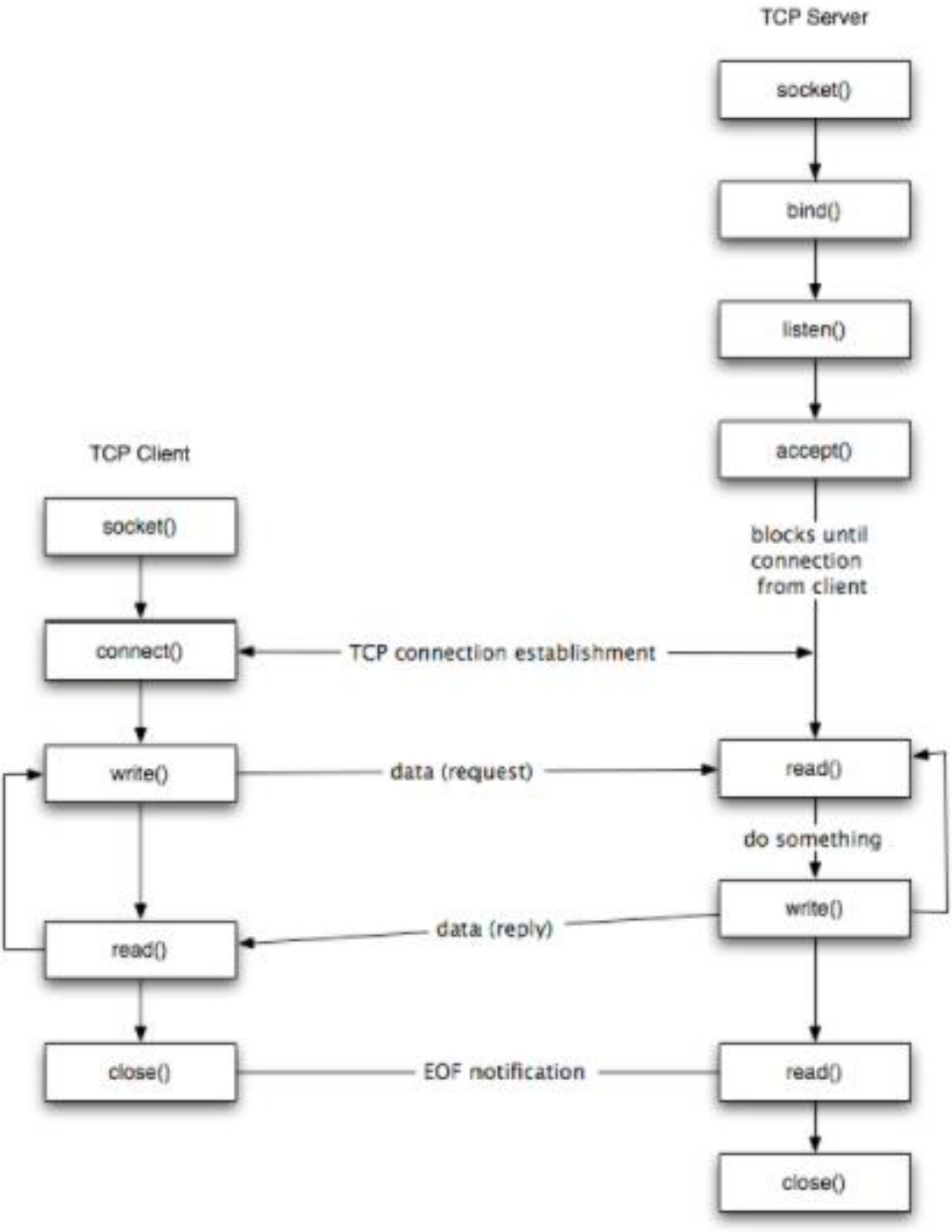
This package consists of the following files,

1. README
2. Makefile
3. Echo.C
4. Echos.C
5. TestCases_Screenshots.pdf

USAGE INSTRUCTIONS:

1) To compile the code, run the Makefile : Command : make
2) Run the server. Command: ./echos PortNumber .
3) Run the client. Command: ./echo IPV4Addr PortNumber
4) To clean the executables, Command: make clean

ARCHITECTURE FOLLOWED:



The above TCP architecture was incorporated in order to develop a simple Echo server and Client. The socket functions used in the implementation of the Echo Server and Client have been mentioned and discussed in detail below.

SERVER SOCKET FUNCTIONS USED:

1) **Socket() :**
   Syntax : **int socket(int** *domain*, **int** *type*, **int** *protocol*);
   Parameters based on defined architecture , parameter 1: domain. This can be either AF_INET or AF_INET6. Since we are using TCP, domain is defined as AF_INET. It can also be defined as AF_UNSPEC.
2) **Bind() :**
   Syntax: **int bind(int** *sockfd*, **const struct sockaddr *** *my_addr* **",** socklen_t " addrlen)
   Bind() is used to bind the socket descriptor. Its parameters include, the socket descriptor which we want to bind, the address structure object and the length of the address Our code uses the sockaddr_in structure to define the socket parameters. The server address from this structure is type casted to struct sockaddr type in order to bind the socket address.
3) **Listen()**
   Syntax : **int listen(int** *sockfd*, **int** *backlog***)**
   The server listens to the incoming client connection requests through listen() call. Backlog mentions the number of clients that could be connected to the server before a timeout. In our code this is defined as 16. This means that after 16 client connections, the server returns a error message when the 17th client tries to connect to the server.
4) **Accept()**
   Syntax: **int accept(int** *sockfd*, **struct sockaddr *** *addr*, **socklen_t *** *addrlen***)**
   The server waits for connection request from client , and accepts a request through this Accept call. It uses the client socket descriptor and address as its parameters. Since, the code tries to establish a multiclient connection, fork is used, which creates the child processes. These child processes connect and transmit data. The parent server process however, stays at the accept call waiting for accepting connections from new clients.
5) **Read()**
   The read function, reads the data from the client socket upto MAX_DATA_SIZE -1 , where MAX_DATA_SIZE is 200 in our code. If the bytes received is a -1, then it is a read data read error. Similarly, when the client disconnects, the read function returns 0, thus indicating that the client has disconnected.
6) **Writen()**
   The server reads the data and echoes back to the client. The server writes the data back to the socket , through a user defined function call Writen(). This function uses the write function to write the data bytes to the client socket. If the number of data bytes sent is returned as <=0 then an error is occurred. It is necessary to check if the error is due to EINTR. If EINTR, the variables are reset and the write is performed again.
7) **Close()**
   Once communication is complete the server and client sockets are closed.

CLIENT SOCKET FUNCTIONS USED:

1) **Socket():**
   As used in server, the server() function creates and returns an integer socket descriptor to the client.
2) **Connect()**
   Once Socket descriptor is created, the client tries to connect to the server HOST IP, through the sever port, which is given as an input to the client function. This returns a positive value if the connect is successful and a negative value if some error had occurred. The input parameters are converted to the information of network layer type and sent as parameters for connection.
3) **fgets()**
   Used to read the standard input from the console.This returns a NULL when end of file or end of MAX_characters in its parameters is reached. This is used to read data from the console iteratively . Further Cntrl+D is detected by feof() which returns a true when end of file occurs.
4) **Writen()**
   User defined function which incorporates write function to send data from client to the server.
5) **Readline()**
   User defined function which is used to read the data received from the server. This in turn calls another user defined function called localRead() which reads character by character in the input using read function. It also handles the EINTR scenario and begins read again if the EINTR is true.
6) **Close()**
   Once communication is complete the server and client sockets are closed.

ERROR HANDLING

1. The writen() and readline functions have incorporated measures for EINTR , where the write or read is started again if it occurs
2. On each function error, the errno, ie the error number is extracted and its corresponding error string is printed.

REFERENCES USED:

- UNP Vol-1, 3rd Edition Stevens et al.
- Beej's Guide to Network Programming