MAJOR ASSIGNMENT-02

UNIX NetworkProgramming (CSE 4042)

TCP Echo Server Design With a Single Process Model

Problem statement

Development of a TCP echo-server as a single process that uses **`select`** to handle any number of clients, instead of **`fork`**ing one child per client.

Design Framework

The following Figure 1 shows the client-server along with the functions used for input and output. It is shown two arrows between the client and server in the figure, but that is really one full-duplex TCP connection.

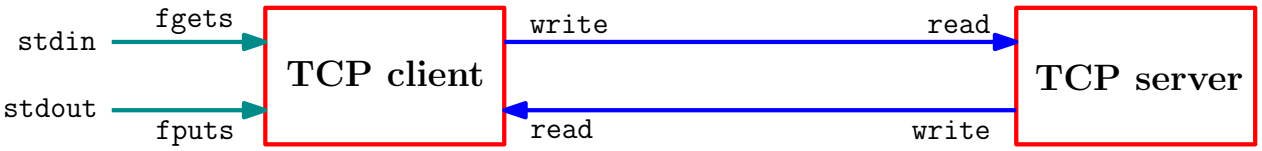

Figure 1: TCP echo client and server

The echo server is to be designed to perform the following steps:

i) The client reads a line of text from its standard input and write the line to the server.

ii) The server reads the line from its network input and echoes the line back to the client.

iii) The client reads the echoed line and prints its on its standard output.

iv) The server will keep a track of the client number that is connected to the server and also the child server process ID.

Design Procedure

Data structures used to keep track of the clients

1. Keep the state of the server as *listening server* before the first client has established a connection as shown in the Figure 2. The server has a single listening descriptor, which is shown as a bullet.

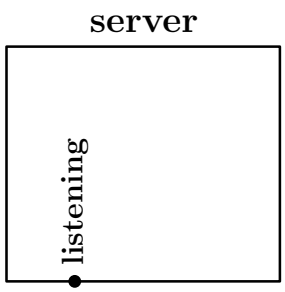

Figure 2: TCP server before first client has established a connection

```
socklen_t clilen,len;
struct sockaddr_in cliaddr, servaddr;
len=sizeof(servaddr);
listenfd=socket(AF_INET,SOCK_STREAM,0);
bind(listenfd,(struct sockaddr *)&servaddr,sizeof(servaddr));
getsockname(listenfd,(struct sockaddr *)&servaddr,&len);
printf("Port for client=%hu\n",ntohs(servaddr.sin_port));
listen(listenfd,15);
```

2. Let the server maintains only a read descriptor set (i.e. **`fd_set rset;`**). The descriptors 0, 1 and 2 are set to standard input, output and error. Therefore, the first availble descriptor for the *listening* socket is 3 as shown in Figure 3. The only nonzero entry in the descriptor set is the entry for the listening sockets and the first argument to **`select()`** will be 4.

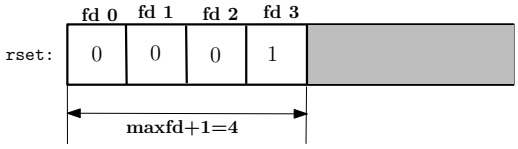

Figure 3: Read descriptor sets for the server

```
fd_set rset,allset;
maxfd=listenfd; /* initialize */
maxi=-1; /* index into client[] array */
FD_ZERO(&allset);
FD_SET(listenfd,&allset);
```

3. Decalre an array of integers named **`client`** of size `FD_SETSIZE` that conatins the connected socket descriptor for each client. Initialize all the elements of that array as -1.

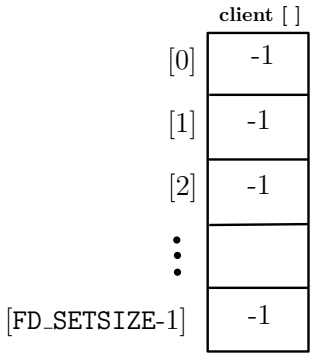

Figure 4: Initial client array in server

```
for(i=0;i<FD_SETSIZE;i++){
   client[i]=-1;
}
```

4. When the first client establishes a connection with the TCP server, the listening socket descriptor becomes readable and the server calls **`accept`**.

```
: : : : : : : :
rset=allset;
nready=select(maxfd+1,&rset,NULL,NULL,NULL);
if(FD_ISSET(listenfd,&rset)){
  clilen=sizeof(cliaddr);
  connfd=accept(listenfd,(struct sockaddr *)&cliaddr,&clilen);
}
: : : : : : : : : :  :
```

The new connected descriptor returned by **`accept`** will be 4. The connection from client to server is shown in the Figure 5.

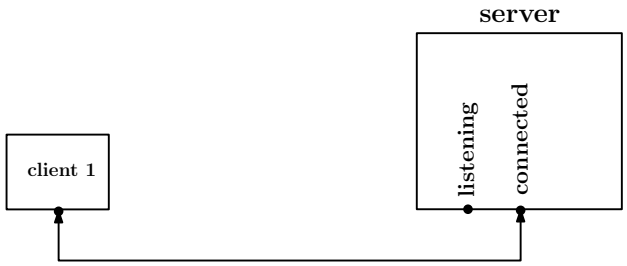

Figure 5: TCP server after first client establishes connection

5. Now, the server must remember the new connected socket in its **`client`** array and the connected socket must be added to the descriptor set as in the Figure 6.

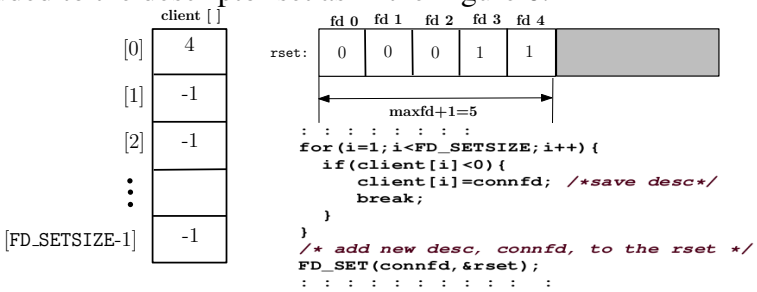

Figure 6: TCP server after first client establishes connection

6. The scenario after the second client establishes a connection to the server as shown in Figure 7.

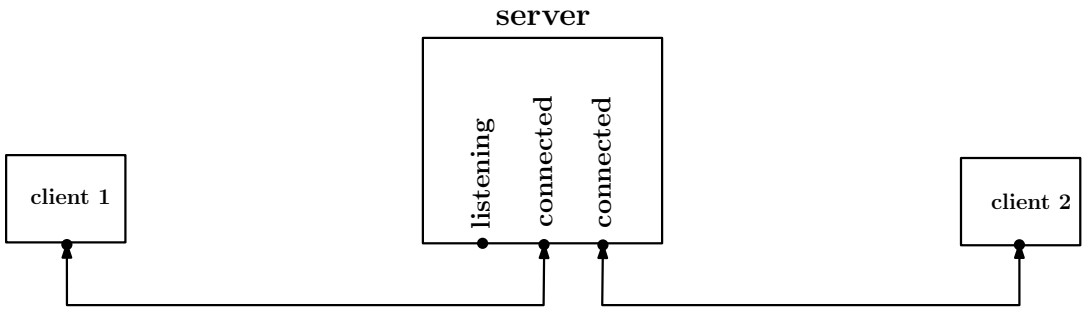

Figure 7: TCP server after second first client connection is established

The new connected socket (assume that it is 5) is remembered, giving the data structures shown in the Figure 8

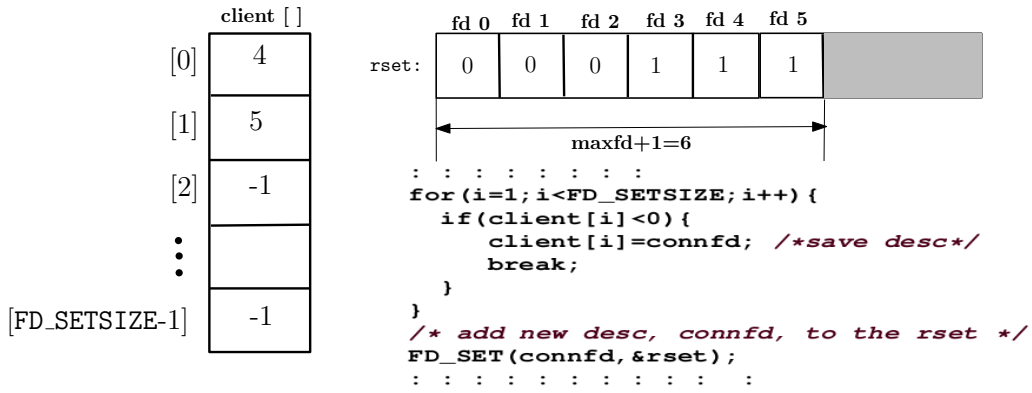

Figure 8: TCP server after first client establishes connection

7. Let us assume that the first client terminates its connection. The client TCP sends a `FIN`, which makes the descriptor 4 in the server is readable. The server reads this connected socket and read returns 0. Now, close this socket and update the client array, **`client[0]=-1`**, along with the descriptor 4 in the descriptor set to 0 as shown in the Figure 9.

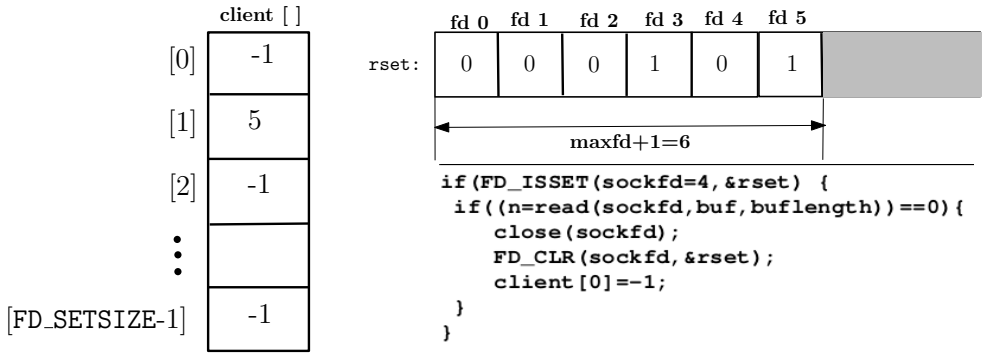

Figure 9: Data structures after the first client terminates its connectiont

Conclusion

Complete the design procedure by adding the other requirements if necessary to meet the problem statement and also run the select loop for infinite long time to provide service to the clients. Additionally, design the appropriate client for this server to communicate. Observe the advanatges and limitations of this kind of server design to handle the clients.

Further Work

You are advised to redesign the TCP echo server using **`poll`** instead of **`select`**. In **`select`**, It is required to allocate a **`client`** array along with a descriptor set. In **`poll`** allocate an array of **`pollfd`** structure to maintain the client information instead of allocation another array. If the **`fd`** member of the **`pollfd`** structure is set to a negative value, then that descriptor is no longer interested. So, the **`events`** member is ignored and the **`revents`** member is set to 0 on return.

You are required to show the sample input and output for various possible cases of this assignment and also the code should able to handle any possible error cases.