SSN College of Engineering, Kalavakkam
Department of Computer Science and Engineering
V Semester - CSE 'B'
UCS1511 NETWORKS LAB

Date :13/08/2020 **Name :** Rahul Ram M
Exercise : 03 **Reg No :** 185001121

CHAT USING TCP

Learning Objective:

To write a socket program to perform chat with multiple clients.

Algorithm for Server:

1. Creating a socket using the function socket(domain, type, protocol) which the returns an integer as the status of the socket creation. Here the domain is AF_INET(iPv4 protocol), type is SOCK_STEAM and protocol as 0.
2. Using bzero(&server_addr, sizeof(server_addr)) function setting values of all the socket structures to null.
3. Using bind() to binf the socket to the address and port number specified in addr(custom data structure). Here, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.
4. listen() function is used to set the server socket in the passive mode, where it waits for the client to approach the server to make a connection, with maximum number of connection in this case is 3.
5. Intitialize all the values of the client_socket array to 0(means we don't have to listen to them).
6. Setting a while loop which runs till server enters 'exit' or terminate using ctrl+z.
   - Clear the list of socket descriptors to monitor using FD_ZERO(&read_fds).
   - Add the descriptor of the server to the list using FD_SET(server_socket, &read_fds).
   - Assign max_sd as server_socket.
   - Using for loop over client_socket array to select valid descriptors , add them to the list of descriptors to monitor and assign the higher number to max_fd.
   - Using select() wait for the activity on one of the sockets in the read_fds indefinetely(timeout is NULL).
   - Using FD_ISSET(server_socket, &read_fds) tests for an incoming connection.
     - If it detects any incoming connection, accept the connection using accept() which creates a socket and assign the new socket any free space in client_socket array.
   - Now loop over all the valid fds in the client_socket array
     - Using FD_ISSET(sd, &read_fds) tests for any message from the descriptor.
     - If the value read by read() on this descriptor is 0, close the socket using close() and reassign the value in the array to be 0(Terminates the connection to that client) for reuse.
     - Else read() has read some message sent by the client to the server.
     - Print the message.

- Read message from server using scanf(" %[^\n]", buffer) and write to the client using ‘write(sd, buffer, sizeof(buffer))’ if the message is not ‘exit’, else terminate the program.

Algorithm for Client(same for all the clients):

1. Creating a socket using the function socket(domain, type, protocol) which the returns an integer as the status of the socket creation. Here the domain is AF_INET(iPv4 protocol), type is SOCK_STEAM and protocol as 0.
2. Using bzero(&server_addr, sizeof(server_addr)) function setting values of all the socket structures to null.
3. The above two steps are same as the server.
4. The connect() system call connects the socket referred to by the file descriptor socket_fd to the address specified by server_addr. Server’s address and port is specified in server_addr.
5. Read the client name from the client using scanf()
6. Setting a while loop which runs till ‘exit’ is given as message.
    1. Clearing the buffer using bzero().
    2. Reading message from the client using ‘scanf(" %[^\n]", message)’ and concatenate name of the client , ‘:’ symbol and message and save it in buffer.
    3. If the message is ‘exit’, close the descriptor using close() and break the loop.
    4. Else write the message in the buffer to the server using write(client_fd, buff, sizeof(buff)).
    5. Now read the message from the server using ‘value = read(client_fd, buff, sizeof(buff))’ and print the message from the server using printf();

Program for Server:

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/time.h>

#define TRUE 1
#define FALSE 0
#define PORT 8080

int main()
{
        int server_socket, addr_size, new_socket, client_socket[10],
                max_clients = 10, activity, val, sd, max_sd, opt = TRUE;

        struct sockaddr_in server_addr;

        char buffer[1024];

        fd_set read_fds;

        for (int i = 0; i < max_clients; i++)
```

```
    {
      client_socket[i] = 0;
    }

      if((server_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
      {
            perror("Socker error");
            exit(1);
      }

      if( setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, (char *)&opt, sizeof(opt))
< 0 )
      {
      perror("setsockopt");
      exit(1);
      }

      server_addr.sin_family = AF_INET;
      server_addr.sin_addr.s_addr = INADDR_ANY;
      server_addr.sin_port = htons(PORT);

      if(bind(server_socket, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
      {
            perror("Bind error: ");
            exit(1);
      }

      if(listen(server_socket,3) < 0)
      {
            perror("Listen error");
            exit(1);
      }

      addr_size = sizeof(server_addr);

      printf("Waiting for client...\n");

      while(TRUE)
      {
            FD_ZERO(&read_fds);

            FD_SET(server_socket, &read_fds);
            max_sd = server_socket;

            for(int i = 0; i < max_clients; i++)
            {
                  sd = client_socket[i];

                  if(sd > 0)
                  {
                        FD_SET(sd, &read_fds);
                  }
```

```
                if(sd > max_sd)
                {
                        max_sd = sd;
                }

        }
        activity = select( max_sd + 1, &read_fds, NULL, NULL, NULL);

        if ((activity < 0) && (errno!=EINTR))
        {
                printf("Select error");
        }

        if (FD_ISSET(server_socket, &read_fds))
        {
                if ((new_socket = accept(server_socket, (struct sockaddr *)&server_addr,
(socklen_t*)&addr_size))<0)
                {
                        perror("Accept error");
                        exit(1);
                }

                for (int i = 0; i < max_clients; i++)
                {
                        if( client_socket[i] == 0 )
                        {
                                client_socket[i] = new_socket;
                                break;
                        }
                }
        }

        for(int i = 0; i < max_clients; i++)
        {
                sd = client_socket[i];
                if (FD_ISSET(sd, &read_fds))
                {

                        if ((val = read(sd , buffer, sizeof(buffer)) == 0))
                        {
                                close(sd);
                                client_socket[i] = 0;
                        }
                        else
                        {
                                printf("%s\n", buffer);
                                bzero(buffer, sizeof(buffer));
                                printf("Server : ");
                                scanf(" %[^\n]", buffer);
                                if(strcmp(buffer, "exit") == 0)
                                {
```

```
                                        printf("Server shutdown successfully!\n");
                                        exit(0);
                                }
                                val = write(sd, buffer, sizeof(buffer));

                        }
                }
        }
    }

    return 0;
}
```

Program for Client(same for all the clients):

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 8080

int main()
{
    int client_fd, value;
    struct sockaddr_in server_addr;

    char buff[1024], name[10], message[1000];

    printf("Connecting to server...\n");

    if((client_fd=socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("Socket error");
    }

    bzero(&server_addr,sizeof(server_addr));

    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    server_addr.sin_port = htons(PORT);

    if(connect(client_fd,(struct sockaddr*)&server_addr, sizeof(server_addr)) != 0)
    {
        perror("Connect error");
    }
    else
    {
        printf("Connected to the Server...\n");
    }
```

```
	printf("Enter client name : ");
	scanf("%s", name);

	while(1)
	{
		bzero(buff, 1024);
		printf("%s : ", name);
		scanf(" %[^\n]", message);
		strcat(buff, name);
		strcat(buff, " : ");
		strcat(buff, message);

		if(strcmp(message, "exit") == 0){
			close(client_fd);
			printf("Disconnected from server...\n");
			break;
		}
		else
		{
			value = write(client_fd, buff, sizeof(buff));
		}
		value = read(client_fd, buff, sizeof(buff));
		printf("Server: %s\n", buff);

	}

	return 0;
}
```

Screenshot for Server:

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/time.h>

#define TRUE 1
#define FALSE 0
#define PORT 8080

int main()
{
	int server_socket, addr_size, new_socket, client_socket[10],
		max_clients = 10, activity, val, sd, max_sd, opt = TRUE;

	struct sockaddr_in server_addr;

	char buffer[1024];

	fd_set read_fds;

	for (int i = 0; i < max_clients; i++)
	{
		client_socket[i] = 0;
	}

	if((server_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
	{
		perror("Socker error");
		exit(1);
	}

	if( setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, (char *)&opt, sizeof(opt)) < 0 )
	{
		perror("setsockopt");
		exit(1);
	}

	server_addr.sin_family = AF_INET;
	server_addr.sin_addr.s_addr = INADDR_ANY;
	server_addr.sin_port = htons(PORT);

	if(bind(server_socket, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
	{
		perror("Bind error: ");
		exit(1);
	}
```

```
48      if(bind(server_socket, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
49      {
50              perror("Bind error: ");
51              exit(1);
52      }
53
54      if(listen(server_socket,3) < 0)
55      {
56              perror("Listen error");
57              exit(1);
58      }
59
60      addr_size = sizeof(server_addr);
61
62      printf("Waiting for client...\n");
63
64      while(TRUE)
65      {
66              FD_ZERO(&read_fds);
67
68              FD_SET(server_socket, &read_fds);
69              max_sd = server_socket;
70
71              for(int i = 0; i < max_clients; i++)
72              {
73                      sd = client_socket[i];
74
75                      if(sd > 0)
76                      {
77                              FD_SET(sd, &read_fds);
78                      }
79
80                      if(sd > max_sd)
81                      {
82                              max_sd = sd;
83                      }
84
85              }
86              activity = select( max_sd + 1, &read_fds, NULL, NULL, NULL);
87
88              if ((activity < 0) && (errno!=EINTR))
89              {
90                      printf("Select error");
91              }
92
93              if (FD_ISSET(server_socket, &read_fds))
94              {
95                      if ((new_socket = accept(server_socket, (struct sockaddr *)&server_addr,
  (socklen_t*)&addr_size))<0)
96                      {
97                              perror("Accept error");
98                              exit(1);
99                      }
```

```
91                }
92
93                if (FD_ISSET(server_socket, &read_fds))
94                {
95                        if ((new_socket = accept(server_socket, (struct sockaddr *)&server_addr,
   (socklen_t*)&addr_size))<0)
96                        {
97                                perror("Accept error");
98                                exit(1);
99                        }
100
101                       for (int i = 0; i < max_clients; i++)
102                       {
103                               if( client_socket[i] == 0 )
104                               {
105                                       client_socket[i] = new_socket;
106                                       break;
107                               }
108                       }
109               }
110
111               for(int i = 0; i < max_clients; i++)
112               {
113                       sd = client_socket[i];
114                       if (FD_ISSET(sd, &read_fds))
115                       {
116
117                               if ((val = read(sd , buffer, sizeof(buffer)) == 0))
118                               {
119                                       close(sd);
120                                       client_socket[i] = 0;
121                               }
122                               else
123                               {
124                                       printf("%s\n", buffer);
125                                       bzero(buffer, sizeof(buffer));
126                                       printf("Server : ");
127                                       scanf(" %[^\n]", buffer);
128                                       if(strcmp(buffer, "exit") == 0)
129                                       {
130                                               printf("Server shutdown successfully!\n");
131                                               exit(0);
132                                       }
133                                       val = write(sd, buffer, sizeof(buffer));
134
135                               }
136                       }
137               }
138       }
139
140       return 0;
141 }
142
```

Screenshot for Client:

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 8080

int main()
{
        int client_fd, value;
        struct sockaddr_in server_addr;

        char buff[1024], name[10], message[1000];

        printf("Connecting to server...\n");

        if((client_fd=socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
                perror("Socket error");
        }

        bzero(&server_addr,sizeof(server_addr));

        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
        server_addr.sin_port = htons(PORT);

        if(connect(client_fd,(struct sockaddr*)&server_addr, sizeof(server_addr)) != 0)
        {
                perror("Connect error");
        }
        else
        {
                printf("Connected to the Server...\n");
        }

        printf("Enter client name : ");
        scanf("%s", name);

        while(1)
        {
                bzero(buff, 1024);
                printf("%s : ", name);
                scanf(" %[^\n]", message);
                strcat(buff, name);
                strcat(buff, " : ");
                strcat(buff, message);

                if(strcmp(message, "exit") == 0){
                        close(client_fd);
                        close(client_fd);
                        printf("Disconnected from server...\n");
                        break;
                }
                else
                {
                        value = write(client_fd, buff, sizeof(buff));
                }
                value = read(client_fd, buff, sizeof(buff));
                printf("Server: %s\n", buff);

        }

        return 0;
}
```

Server Output:

```
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_03$ ./s
Waiting for client...
Client1 : hello server
Server : name please?
Client1 : Client 1
Server : hello client1
Client1 : hi
Server : Welcome to socket programming
Client2 : this is client2
Server : Welcome client 2
Client1 : bye gg
Server : ok bye
Client2 : Going to sleep
Server : exit
Server shutdown successfully!
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_03$
```

Client1 Output:

```
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_03$ ./ca
Connecting to server...
Connected to the Server...
Enter client name : Client1
Client1 : hello server
Server: name please?
Client1 : Client 1
Server: hello client1
Client1 : hi
Server: Welcome to socket programming
Client1 : bye gg
Server: ok bye
Client1 : exit
Disconnected from server...
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_03$
```

Client2 Output:

```
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_03$ ./cb
Connecting to server...
Connected to the Server...
Enter client name : Client2
Client2 : this is client2
Server: Welcome client 2
Client2 : Going to sleep
Server: Client2 : Going to sleep
Client2 : exit
Disconnected from server...
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_03$
```

Learning Outcomes:

This assignment helped me to

1. Write program for server and client with socket programming.
2. Understand various functions invloved in creating, estabilishing, maintaining, Sending, recieving and termininating the connection between the server and client.
3. Connect multiple clients to the server using select() system call.
4. Learn about the data structures used for select() :fd_set.
5. Write code to make server and client communicate with each other using read() and write() functions.