
Assignment 3 – Multi-user Chat using UDP

Date: 02/09/2022

Aim:

Develop a chat application between a client and server using UDP. Update the program to support multiple clients.

Algorithm:

Server:

1. Start
2. Create a UDP socket using socket() system call.
3. The bind() system call is used to bind the socket with a specified address defined by sockaddr_in pointer, with the address, family, port set accordingly.
4. The bzero() system call is used to clear the address pointer initially.
5. Initialize a descriptor set for select() and calculate a maximum of 3 descriptors for which the server will wait.
6. Call select() and get the ready descriptor (UDP).
7. Handle the new connection and receive the data gram.
8. Stop

Client:

1. Start
2. Create a UDP socket using socket() system call.
3. Send a message to the server.
4. Wait until a response from the server is received.
5. Close socket descriptor and exit.
6. Stop

Code:

```
//Server for multi-client chat implementation using UDP
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netdb.h>

#define MAX 256
#define CLIENT 5
#define SA struct sockaddr


int fd_max(const int arr[CLIENT], int sock_fd, fd_set *readfds)
{
   //Finding the largest file descriptor, required for select()

   int max = -1;
   for (int i = 0; i < CLIENT; ++i)
   {
      max = (arr[i] > max) ? arr[i] : max;

      // If it is valid, add it to set of descriptors
      if (arr[i] > 0)
         FD_SET(arr[i], readfds);
   }

   max = (sock_fd > max) ? sock_fd : max;

   return max + 1;
}


int main(int argc, char **argv)
{
   if(argc < 2){
      fprintf(stderr, "Please pass port number as second argument!\n");
      exit(EXIT_FAILURE);
```

```
}

int PORT = atoi(argv[1]);

int sockfd, new_fd[CLIENT] = {0}, len;
struct sockaddr_in servaddr, cli;
char buff[MAX];


// Setting timeouts for select()
struct timeval tv;
tv.tv_sec = 1;
tv.tv_usec = 0;

// Tracking activity on descriptors
fd_set readfds;


// socket create and verification
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if (sockfd == -1)
{
   printf("---Socket creation: failed---\n");
   exit(EXIT_FAILURE);
}
else
   printf("---Socket creation: successful---\n");

bzero(&servaddr, sizeof(servaddr));
memset(&servaddr, 0, sizeof(servaddr));
memset(&cli, 0, sizeof(cli));

// assign IP, PORT
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(PORT);

// Binding newly created socket to given IP and verification
if ((bind(sockfd, (SA *)&servaddr, sizeof(servaddr))) != 0)
{
   printf("---Socket bind: failed---\n\n");
   exit(EXIT_FAILURE);
}
```

```
   else
      printf("---Socket bind: successful---\n\n");

   len = sizeof(cli);

   bzero(new_fd, sizeof(int) * CLIENT);

   while (1)
   {
      FD_ZERO(&readfds);
      FD_SET(sockfd, &readfds);

      // New connection detected
      if (FD_ISSET(sockfd, &readfds))
      {
         int client = recvfrom(sockfd, buff, MAX, MSG_WAITALL, (struct sockaddr *)&cli,
&len);
         if (client < 0)
         {
            fprintf(stderr, "Accept error!\n");
            exit(1);
         }

         for (int i = 0; i < CLIENT; i++)
            if (new_fd[i] == 0)
            {
               new_fd[i] = client;
               break;
            }
         FD_CLR(sockfd, &readfds);
      }


      int limit = fd_max(new_fd, sockfd, &readfds);

      for (int i = 0; i < CLIENT; i++)
      {
         if (new_fd[i] < 0)
            continue;

         // Message from client
         if (FD_ISSET(new_fd[i], &readfds))
         {
```

```
            int count = read(new_fd[i], buff, MAX);

            // Client has terminated
            if (strcmp(buff, "END") == 0)
            {
               close(new_fd[i]);
               new_fd[i] = 0;
               printf("Client %d disconnected!\n", i);
            }

            else
            {
               printf("Message from client %d: %s \n", (i+1), buff);
               bzero(buff, MAX);
               printf("\tServer to client: ");
               scanf("%[^\n]", buff);
               getchar();

               if(strcmp(buff,"KILL") == 0) exit(EXIT_SUCCESS);

               // Write response to client
               write(new_fd[i], buff, MAX);
               sendto(sockfd, (char *) buff, strlen(buff), MSG_CONFIRM, (struct sockaddr*)&cli,
len);
            }

         }
      }
   }

   printf("Server Exit!\n");

   // Close the socket
   close(sockfd);

   return 0;
}


//Client for multi-client chat implementation using UDP
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <string.h>

#define MAX 256
#define SA struct sockaddr

int main(int argc, char **argv)
{
   if(argc < 2){
      fprintf(stderr, "Please pass port number of server as second argument!\n");
      exit(EXIT_FAILURE);
   }

   int PORT = atoi(argv[1]);
   int sockfd, connfd;
   int size;
   struct sockaddr_in servaddr, cli;
   char buff[MAX];
   int n, len; // socket create and verification
   sockfd = socket(AF_INET, SOCK_DGRAM, 0);
   if (sockfd == -1)
   {
      printf("---Socket creation: failed---\n");
      exit(0);
   }
   else
      printf("---Socket creation: successful---\n");

   bzero(&servaddr, sizeof(servaddr));
   // assign IP, PORT
   servaddr.sin_family = AF_INET;
   servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
   servaddr.sin_port = htons(PORT);

   // connect the client socket to server socket
   if (connect(sockfd, (SA *)&servaddr, sizeof(servaddr)) != 0)
   {
      printf("---Connection with Server: failed---\n\n");
      exit(0);
   }
```

```
   else
                printf("---Connection with Server: successful---\n\n");

   len = sizeof(servaddr);

   int temp = 0;

   while (1)
   {
      bzero(buff, MAX);
      printf("\tClient to server: ");
      scanf("%[^\n]s",buff);
      getchar();
      write(sockfd, buff, MAX);
      sendto(sockfd, (char *) buff, strlen(buff), MSG_CONFIRM, (struct sockaddr*)&servaddr,
len);

      if(strcmp(buff, "END") == 0) break;

      read(sockfd, buff, MAX);
      n = recvfrom(sockfd, (char *) buff, MAX, MSG_WAITALL, (struct sockaddr*)&servaddr,
&len);
      buff[n] = '\0';
      printf("Message from server: %s\n", buff);
   }
   printf("Client Exit!\n");
   close(sockfd);
}
```

Output:

Server side:

```
kri@Krithika-PC-Win11:/mnt/e/code$ ./server 8080
---Socket creation: successful---
---Socket bind: successful---

Message from client 59934: Hi!
        Server to client: Hello

Message from client 46539: Hey
        Server to client: Hey you!

Message from client 59934: This is client 1.
        Server to client: Alright

Message from client 46539: This is client 2.
        Server to client: Ok

Message from client 46539: Gtg
        Server to client: Cool

Message from client 59934: Same here. Pls exit chat.
        Server to client: exit

Server Exit...
kri@Krithika-PC-Win11:/mnt/e/code$
```

Client-1 side:

```
kri@Krithika-PC-Win11:/mnt/e/code$ ./client 8080
---Socket creation: successful---
---Connection with Server: successful---

        Client to server: Hi!

Message from server: Hello
        Client to server: This is client 1.

Message from server: Alright
        Client to server: Same here. Pls exit chat.

Message from server: exit

Client Exit...
kri@Krithika-PC-Win11:/mnt/e/code$
```

Client-2 side:

```
kri@Krithika-PC-Win11:/mnt/e/code$ ./client2 8080
---Socket creation: successful---
---Connection with Server: successful---

        Client to server: Hey

Message from server: Hey you!
        Client to server: This is client 2.

Message from server: Ok
        Client to server: Gtg

Message from server: Cool
```

Learning outcomes:

- The process of establishing a connection between a client and a server using UDP was understood.
- Socket programming using UDP was understood and implemented.
- A multi-user chat with a single server and multiple clients was implemented.
