Name: Shubhan Singh

SE comps B/Batch C

2022300118

CCN Exp 2

Aim: Network Socket Programming

Part-1: Implement the following rudimentary string processing application using connectionoriented client-server programming. Some guidelines for the implementation are as follows. The client will send a textual paragraph terminated by '\n' to the server (assume that in the paragraph, '.' appears only at the end of sentences and nowhere else). The server will compute the number of characters, number of words, and number of sentences in the paragraph, and send these numbers back to the client. The client will print these numbers on the screen.

Source code: (C language)

Server code:

```
// Server side C program to demonstrate Socket
```

```c
// programming
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include <asm-generic/socket.h>
#define PORT 8080

char* process_response(char* buffer){
    char* response=malloc(1024*sizeof(char));
    int nochar=strlen(buffer);
    int nowords=0,nosentences=0;
    if(nochar<=0){
        goto end;
    }
    for(int i=0;i<nochar;i++){
        while(buffer[i]!=' ' && buffer[i]!='.'){
            if(i==nochar){
                break;
            }
            i++;
        }
        nowords++;
    }
    for(int i=0;i<nochar-1;i++){
```

```
        while(buffer[i]!='.'){
            if(i==nochar){
                break;
            }
            i++;
        }
        nosentences++;
    }
    end:
    snprintf(response,1023,"\nNo. of characters: %d\nNo.of words: %d\nNo. of sentences:
%d\n",nochar,nowords,nosentences);
    return response;
}

int main(int argc, char const *argv[])
{
    int server_fd, new_socket;
    ssize_t valread;
    struct sockaddr_in address;
    int opt = 1;
    socklen_t addrlen = sizeof(address);
    char buffer[1024] = {0};
    char *hello = "Hello from server";

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
```

```
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Forcefully attaching socket to the port 8080
    if (setsockopt(server_fd, SOL_SOCKET,SO_REUSEADDR | SO_REUSEPORT, &opt,sizeof(opt)))
    {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Forcefully attaching socket to the port 8080
    if (bind(server_fd, (struct sockaddr *)&address,sizeof(address)) < 0)
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    if (listen(server_fd, 3) < 0)
    {
        perror("listen");
        exit(EXIT_FAILURE);
    }
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address, &addrlen)) < 0)
    {
```

```
        perror("accept");
        exit(EXIT_FAILURE);
    }
    valread = read(new_socket, buffer,1024 - 1); // subtract 1 for the null terminator at the end
    printf("received message: %s\n", buffer);
    int i=0;
    while(buffer[i]==' '){//stripping any leading whitespaces
        i++;
    }
    char* response=process_response(buffer+i);
    send(new_socket, response, strlen(response), 0);
    printf("response sent\n");
    free(response);
    // closing the connected socket
    close(new_socket);
    // closing the listening socket
    close(server_fd);
    return 0;
}
```

Client code:

```
// Client side C program to demonstrate Socket
// programming
#include <arpa/inet.h>
```

```c
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 8080

int main(int argc, char const* argv[])
{
    int status, valread, client_fd;
    struct sockaddr_in serv_addr;
    char sentence[1024]={0};
    char buffer[1024] = { 0 };
    if ((client_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary
    // form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<= 0) {
        printf(
            "\nInvalid address/ Address not supported \n");
        return -1;
    }
```

```
    if ((status= connect(client_fd, (struct sockaddr*)&serv_addr,sizeof(serv_addr)))< 0) {
        printf("\nConnection Failed \n");
        return -1;
    }
    printf("Enter paragraph:\n");
    scanf("%[^\n]",sentence);
    send(client_fd, sentence, strlen(sentence), 0);
    printf("Message sent\n");
    valread = read(client_fd, buffer,1024 - 1); // subtract 1 for the null terminator at the end
    printf("%s\n", buffer);

    // closing the connected socket
    close(client_fd);
    return 0;
}
```

Output:

```
shubhan@MSI: ~/programs/C
shubhan@MSI:~$ cd programs/CCN
shubhan@MSI:~/programs/CCN$ ls
chat_client.c  client.c                  multiple
chat_server    client.c:Zone.Identifier  server
chat_server.c  constants.h               server.c
chclient       multi                     server.c:Zone.Identifier
client         multi.c
shubhan@MSI:~/programs/CCN$ ./server
received message: Implement the following rudimentary string processing
application using connectionoriented client-server programming. Some gui
delines for the implementation are as follows. The client will send a te
xtual paragraph terminated by '\n' to the server (assume that in the par
agraph, '.' appears only at the end of sentences and nowhere else). The
server will compute the number of characters, number of words, and numbe
r of sentences in the paragraph, and send these numbers back to the clie
nt. The client will print these numbers on the screen.
response sent
shubhan@MSI:~/programs/CCN$
```

```
shubhan@MSI: ~/programs/C
shubhan@MSI:~$ cd programs/CCN
shubhan@MSI:~/programs/CCN$ ./client
Enter paragraph:
Implement the following rudimentary string processing application using connecti
onoriented client-server programming. Some guidelines for the implementation are
 as follows. The client will send a textual paragraph terminated by '\n' to the
server (assume that in the paragraph, '.' appears only at the end of sentences a
nd nowhere else). The server will compute the number of characters, number of wo
rds, and number of sentences in the paragraph, and send these numbers back to th
e client. The client will print these numbers on the screen.
Message sent

No. of characters: 548
No.of words: 88
No. of sentences: 6
shubhan@MSI:~/programs/CCN$
```

Part 2: Make it concurrent so that it can serve multiple clients at a time. (Multiple clients on multiple terminals and single server terminals)

Source code(C language):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>

#define MAX_CLIENTS 10
#define BUFFER_SIZE 1024

// Function to process the paragraph and compute the required statistics
char* processParagraph(char* buffer) {
    char* response=malloc(1024*sizeof(char));
    int nochar=strlen(buffer);
    int nowords=0,nosentences=0;
    if(nochar<=0){
        goto end;
    }
    for(int i=0;i<nochar;i++){
        if(buffer[i]==' ' || buffer[i]=='.'){
            nowords++;
        }
    }
    for(int i=0;i<nochar;i++){
```

```
        if(buffer[i]=='.'){
            nosentences++;
        }
    }
    end:
    snprintf(response,1023,"\nNo. of characters: %d\nNo.of words: %d\nNo. of sentences:
%d\n",nochar,nowords,nosentences);
    return response;
}

// Function to handle each client connection
void *handleClient(void *arg) {
    int clientSocket = *((int *)arg);
    free(arg);

    char buffer[BUFFER_SIZE];

    while (1) {
        // Receive paragraph from the client
        ssize_t bytesRead = recv(clientSocket, buffer, sizeof(buffer), 0);

        if (bytesRead <= 0) {
            break; // Connection closed or error
        }

        buffer[bytesRead] = '\0';
```

```
        // Check if the client wants to close the connection
        if (strcmp(buffer, "exit\n") == 0) {
            break;
        }

        // Send the results back to the client
        char* response=processParagraph(buffer);

        send(clientSocket, response, strlen(response), 0);
    }

    // Close the client socket
    close(clientSocket);
    return NULL;
}

int main() {
    int serverSocket, clientSocket;
    struct sockaddr_in serverAddr, clientAddr;
    socklen_t clientAddrLen = sizeof(clientAddr);

    // Create socket
    serverSocket = socket(AF_INET, SOCK_STREAM, 0);

    // Set up server address struct
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(8888); // Port number
```

```c
    serverAddr.sin_addr.s_addr = INADDR_ANY;

    // Bind socket to the address
    bind(serverSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr));

    // Listen for incoming connections
    listen(serverSocket, MAX_CLIENTS);

    printf("Server listening on port 8888...\n");

    while (1) {
        // Accept a client connection
        clientSocket = accept(serverSocket, (struct sockaddr *)&clientAddr, &clientAddrLen);

        // Create a thread to handle the client
        pthread_t thread;
        int *clientSocketPtr = malloc(sizeof(int));
        *clientSocketPtr = clientSocket;
        pthread_create(&thread, NULL, handleClient, clientSocketPtr);

        // Detach the thread to avoid memory leaks
        pthread_detach(thread);
    }

    // Close the server socket
    close(serverSocket);
```

```
    return 0;
}
```

Output:

```
shubhan@MSI: ~/programs/C
shubhan@MSI:~$ cd programs/CCN
shubhan@MSI:~/programs/CCN$ gcc multi.c -o multi
shubhan@MSI:~/programs/CCN$ ./multi
Server listening on port 8888...
```

```
shubhan@MSI: ~
shubhan@MSI:~$ telnet localhost 8888
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
My name is shubhan

No. of characters: 20
No.of words: 3
No. of sentences: 0
```

```
shubhan@MSI: ~
shubhan@MSI:~$ telnet localhost 8888
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
My name is shubhan.

No. of characters: 21
No.of words: 4
No. of sentences: 1
```

```
shubhan@MSI: ~
shubhan@MSI:~$ telnet localhost 8888
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
my name is shuhban. this is my CCN submission.

No. of characters: 48
No.of words: 10
No. of sentences: 2
```

23°C Haze | Search | ENG IN | 00:32 31-01-2024

Part-3: Write client-server application for chat server. The two clients connected to the same server should be able to communicate with each other. Communication should be interactive and go on till one of them terminates.

Source code:

Server code:

```
// server.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>
#include "constants.h"

typedef struct {
    int socket;
    struct sockaddr_in address;
} ClientInfo;

ClientInfo clients[10];
int clientCount = 0;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void send_private_message(int sender_index, int recipient_index, const char *message) {
    char private_message[BUFFER_SIZE];
    snprintf(private_message, sizeof(private_message), "[Private from %s:%d]: %s",
             inet_ntoa(clients[sender_index].address.sin_addr),
```

```
            ntohs(clients[sender_index].address.sin_port), message);

    send(clients[recipient_index].socket, private_message, strlen(private_message), 0);
}

void send_group_message(int sender_index, const char *message) {
    char group_message[BUFFER_SIZE];
    snprintf(group_message, sizeof(group_message), "[Group from %s:%d]: %s",
             inet_ntoa(clients[sender_index].address.sin_addr),
             ntohs(clients[sender_index].address.sin_port), message);

    pthread_mutex_lock(&lock);
    for (int i = 0; i < clientCount; ++i) {
        if (i != sender_index) {
            send(clients[i].socket, group_message, strlen(group_message), 0);
        }
    }
    pthread_mutex_unlock(&lock);
}

void *handle_client(void *client_socket) {
    int current_index = clientCount++;
    int socket = *((int *)client_socket);
    char buffer[BUFFER_SIZE];

    while (1) {
        int received_bytes = recv(socket, buffer, sizeof(buffer), 0);
```

```
        if (received_bytes <= 0) {
            break;
        }

        buffer[received_bytes] = '\0';

        if (buffer[0] == '@') {
            // Private message
            int recipient_index;
            char recipient_address[20];
            char private_message[BUFFER_SIZE];

            sscanf(buffer, "@(%19[^:]): %[^\n]", recipient_address, private_message);

            for (int i = 0; i < clientCount; ++i) {
                if (strcmp(inet_ntoa(clients[i].address.sin_addr), recipient_address) == 0) {
                    recipient_index = i;
                    break;
                }
            }

            send_private_message(current_index, recipient_index, private_message);
        } else {
            // Group message
            send_group_message(current_index, buffer);
        }
    }
```

```
    close(socket);

    pthread_mutex_lock(&lock);
    printf("Connection from %s:%d closed.\n",
           inet_ntoa(clients[current_index].address.sin_addr),
           ntohs(clients[current_index].address.sin_port));
    clientCount--;

    // Shift remaining clients to fill the gap
    for (int i = current_index; i < clientCount; ++i) {
        clients[i] = clients[i + 1];
    }
    pthread_mutex_unlock(&lock);

    return NULL;
}

int main() {
    int server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        perror("Error creating server socket");
        exit(EXIT_FAILURE);
    }

    struct sockaddr_in server_address;
    server_address.sin_family = AF_INET;
```

```
    server_address.sin_addr.s_addr = inet_addr(SERVER_ADDRESS);
    server_address.sin_port = htons(SERVER_PORT);

    if (bind(server_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    if (listen(server_socket, 5) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    printf("Server listening on %s:%d\n", SERVER_ADDRESS, SERVER_PORT);

    while (1) {
        struct sockaddr_in client_address;
        int client_socket, client_address_size = sizeof(client_address);
        client_socket = accept(server_socket, (struct sockaddr *)&client_address, (socklen_t
*)&client_address_size);

        pthread_t client_thread;
        int *client_socket_ptr = malloc(sizeof(int));
        *client_socket_ptr = client_socket;

        pthread_create(&client_thread, NULL, handle_client, (void *)client_socket_ptr);
        pthread_detach(client_thread);
```

```
        pthread_mutex_lock(&lock);
        clients[clientCount].socket = client_socket;
        clients[clientCount].address = client_address;
        pthread_mutex_unlock(&lock);
    }

    close(server_socket);

    return 0;
}
```

Client code:

```
// client.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include "constants.h"
#include <pthread.h>

void *receive_messages(void *client_socket) {
    int socket = *((int *)client_socket);
    char buffer[BUFFER_SIZE];
```

```c
    while (1) {
        int received_bytes = recv(socket, buffer, sizeof(buffer), 0);
        if (received_bytes <= 0) {
            break;
        }

        buffer[received_bytes] = '\0';
        printf("%s\n", buffer);
    }

    return NULL;
}

int main() {
    int client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket == -1) {
        perror("Error creating client socket");
        exit(EXIT_FAILURE);
    }

    struct sockaddr_in server_address;
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = inet_addr(SERVER_ADDRESS);
    server_address.sin_port = htons(SERVER_PORT);

    if (connect(client_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
        perror("Connection failed");
```

```
        exit(EXIT_FAILURE);
    }

    pthread_t receive_thread;
    pthread_create(&receive_thread, NULL, receive_messages, (void *)&client_socket);

    char message[BUFFER_SIZE];

    while (1) {
        fgets(message, sizeof(message), stdin);
        send(client_socket, message, strlen(message), 0);
    }

    close(client_socket);

    return 0;
}
```

Header:

```
// constants.h


#ifndef CONSTANTS_H
#define CONSTANTS_H

#define BUFFER_SIZE 1024
```

```
#define SERVER_ADDRESS "127.0.0.1"
#define SERVER_PORT 12345

#endif // CONSTANTS_H
```

Output:

```
shubhan@MSI:~$ cd programs/CCN
shubhan@MSI:~/programs/CCN$ ls
chat_client.c  client                    multi     server.c:Zone.Identifier
chat_server    client.c                  multi.c
chat_server.c  client.c:Zone.Identifier  server
chclient       constants.h               server.c
shubhan@MSI:~/programs/CCN$ ./chat_server
Server listening on 127.0.0.1:12345
Connection from 127.0.0.1:40068 closed.
```

```
shubhan@MSI:~$ telnet localhost 12345
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Hi, this is client 1.
[Group from 127.0.0.1:39552]: this is client 2.
[Group from 127.0.0.1:40068]: this is client 3
[Group from 127.0.0.1:39552]: client 1 is top right
client 2 is bottom left
[Group from 127.0.0.1:40068]: client 3
[Group from 127.0.0.1:40068]: [Group from 127.0.0.1:39552]:
```

```
shubhan@MSI:~$ telnet localhost 12345
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[Group from 127.0.0.1:39862]: Hi, this is client 1.
this is client 2.
[Group from 127.0.0.1:40068]: this is client 3
client 1 is top right
[Group from 127.0.0.1:39862]: client 2 is bottom left
[Group from 127.0.0.1:40068]: client 3
[Group from 127.0.0.1:40068]: [Group from 127.0.0.1:39862]:
```

```
shubhan@MSI:~$ telnet localhost 12345
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[Group from 127.0.0.1:39862]: Hi, this is client 1.
[Group from 127.0.0.1:39552]: this is client 2.
this is client 3
[Group from 127.0.0.1:39552]: client 1 is top right
[Group from 127.0.0.1:39862]: client 2 is bottom left
client 3
^Z[Group from 127.0.0.1:39552]: ^]
telnet> q
Connection closed.
shubhan@MSI:~$
```

(Closing one connection closes the entire server)