**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**

**// 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>**

**#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:**

**A screenshot of a computer

Description automatically generated**

**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**

**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);**

**}**

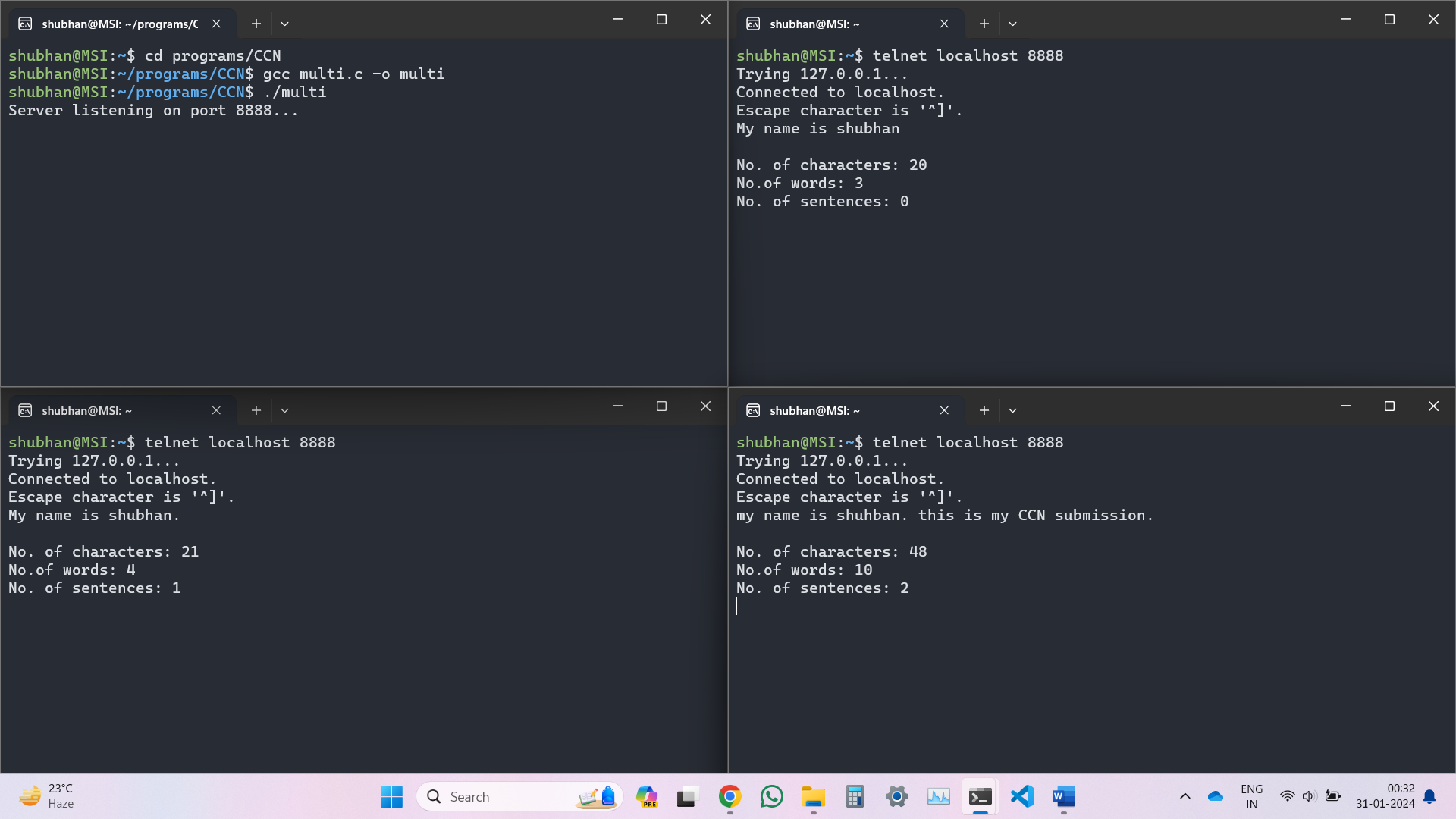
**// Close the server socket**

**close(serverSocket);**

**return 0;**

**}**

**Output:**



**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];**

**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:**

A screenshot of a computer screen

Description automatically generated

(Closing one connection closes the entire server)