**Texas A&M University**

**ECEN 602 600: COMPUTER COMM & NET**

**Machine Problem-1**

**By: Chetan Sai Borra (UIN: 436000947)**

**Ganesh Vairavan Arumugam (UIN: 136004789)**

**Test Cases Report:**

1. **line of text terminated by a newline:**

We use “\n” to mark the end of a line and begin the next message on a new line.

**At server side:**

A computer screen with blue text

Description automatically generated

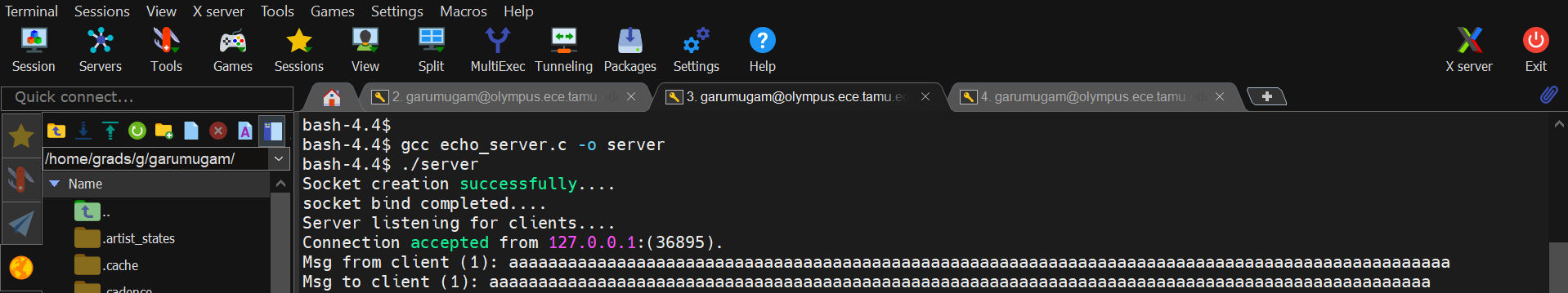
**At Client side:**

**A computer screen with text on it

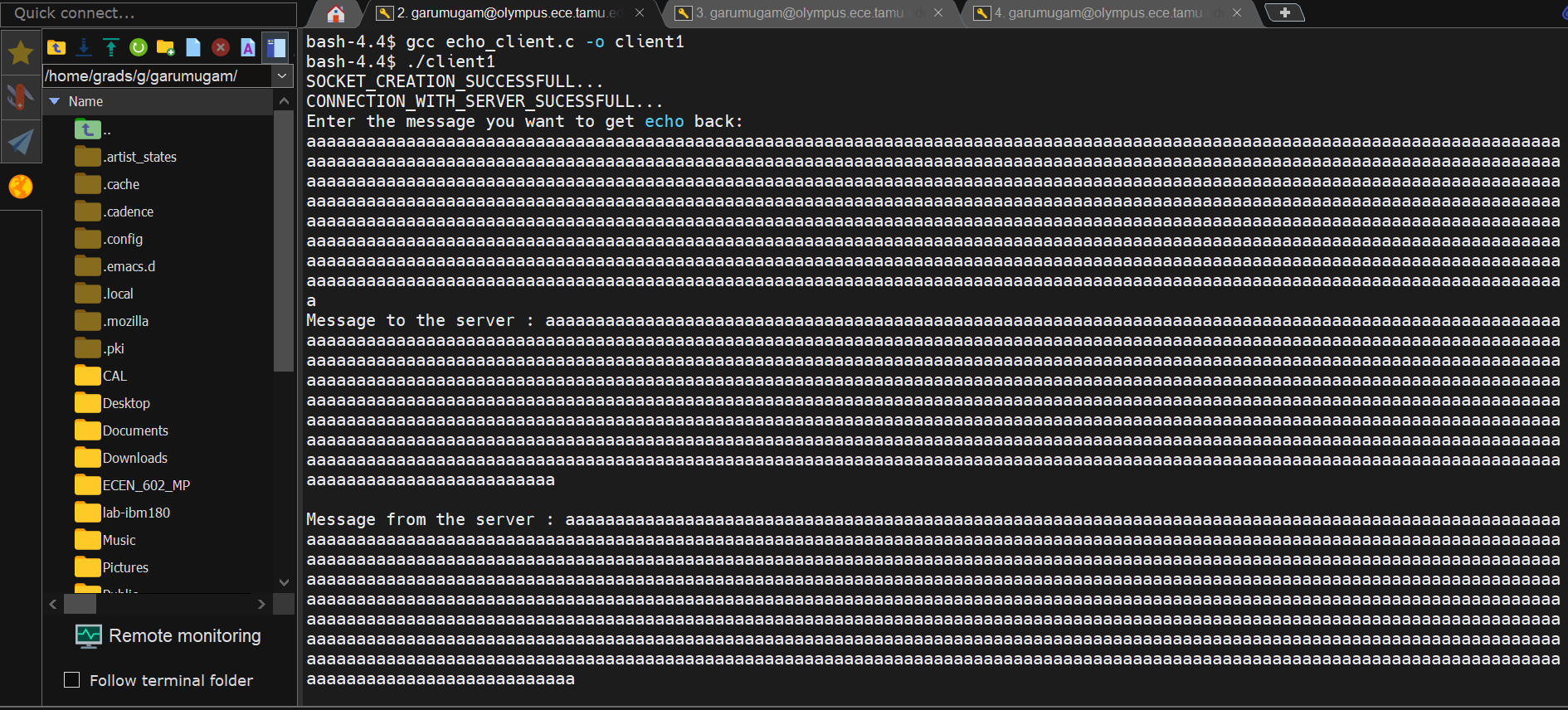
Description automatically generated**

1. **line of text the maximum line length without a newline**

**At Server Side**

****

**At Client side**



1. **line with nocharacters and EOF:**

It is possible to send the message with no characters and EOF.

**At Server Side:**

A screen shot of a computer

Description automatically generated

**At Client Side:**

A screen shot of a computer

Description automatically generated

1. **client terminated after entering text:**

The Clients are terminated after entering the following text in there message **“exit”.**

**At Server Side:**

A computer screen with white text

Description automatically generated

**At Client Side:**

A screen shot of a computer screen

Description automatically generated

1. **Three clientsconnected to the server:**

The server can connect to different clients simultaneously and echo the clients simultaneously.

**At Server Side:**

A screenshot of a computer program

Description automatically generated

**All the clients:**

A screenshot of a computer screen

Description automatically generated

The other versions of the code given by ChatGPT.

**C Source Code**

1. **ChatGPT version code**

**Echo Server code**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <pthread.h>

#define BUFFER\_SIZE 1024

void \*handle\_client(void \*arg) {

int client\_socket = \*(int \*)arg;

free(arg);

char buffer[BUFFER\_SIZE];

int bytes\_read;

while ((bytes\_read = read(client\_socket, buffer, BUFFER\_SIZE)) > 0) {

write(client\_socket, buffer, bytes\_read);

}

close(client\_socket);

return NULL;

}

int main(int argc, char \*argv[]) {

if (argc != 2) {

fprintf(stderr, "Usage: %s <Port>\n", argv[0]);

exit(EXIT\_FAILURE);

}

int server\_socket, \*client\_socket;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t client\_addr\_len = sizeof(client\_addr);

pthread\_t tid;

server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_socket < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(atoi(argv[1]));

if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Bind failed");

close(server\_socket);

exit(EXIT\_FAILURE);

}

if (listen(server\_socket, 10) < 0) {

perror("Listen failed");

close(server\_socket);

exit(EXIT\_FAILURE);

}

printf("Server listening on port %s\n", argv[1]);

while (1) {

client\_socket = malloc(sizeof(int));

\*client\_socket = accept(server\_socket, (struct sockaddr \*)&client\_addr, &client\_addr\_len);

if (\*client\_socket < 0) {

perror("Accept failed");

free(client\_socket);

continue;

}

pthread\_create(&tid, NULL, handle\_client, client\_socket);

pthread\_detach(tid);

}

close(server\_socket);

return 0;

}

**Echo Client code**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define BUFFER\_SIZE 1024

int main(int argc, char \*argv[]) {

if (argc != 3) {

fprintf(stderr, "Usage: %s <IP Address> <Port>\n", argv[0]);

exit(EXIT\_FAILURE);

}

int client\_socket;

struct sockaddr\_in server\_addr;

char buffer[BUFFER\_SIZE];

client\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (client\_socket < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(atoi(argv[2]));

if (inet\_pton(AF\_INET, argv[1], &server\_addr.sin\_addr) <= 0) {

perror("Invalid address");

close(client\_socket);

exit(EXIT\_FAILURE);

}

if (connect(client\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Connection failed");

close(client\_socket);

exit(EXIT\_FAILURE);

}

while (fgets(buffer, BUFFER\_SIZE, stdin) != NULL) {

write(client\_socket, buffer, strlen(buffer));

int bytes\_read = read(client\_socket, buffer, BUFFER\_SIZE);

if (bytes\_read > 0) {

buffer[bytes\_read] = '\0';

printf("Echo: %s", buffer);

}

}

close(client\_socket);

return 0;

}

1. **ChatGPT Enhance version code**

**Echo Server code**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <signal.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

void handle\_client(int client\_socket, int client\_count) {

char buffer[BUFFER\_SIZE];

ssize\_t bytes\_read;

while (1) {

memset(buffer, 0, BUFFER\_SIZE);

bytes\_read = read(client\_socket, buffer, sizeof(buffer));

if (bytes\_read <= 0) {

break;

}

printf("Msg from client (%d): %s\n", client\_count, buffer);

write(client\_socket, buffer, bytes\_read);

if (strncmp("exit", buffer, 4) == 0) {

printf("Server Exit...\n");

break;

}

}

close(client\_socket);

}

int main() {

int server\_socket, client\_socket;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t client\_addr\_len = sizeof(client\_addr);

pid\_t childpid;

int client\_count = 0;

signal(SIGCHLD, SIG\_IGN); // Prevent zombie processes

server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_socket < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

printf("Socket creation successful.\n");

memset(&server\_addr, 0, sizeof(server\_addr));

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 failed");

close(server\_socket);

exit(EXIT\_FAILURE);

}

printf("Socket bind successful.\n");

if (listen(server\_socket, 10) < 0) {

perror("Listen failed");

close(server\_socket);

exit(EXIT\_FAILURE);

}

printf("Server listening on port %d...\n", PORT);

while (1) {

client\_socket = accept(server\_socket, (struct sockaddr\*)&client\_addr, &client\_addr\_len);

if (client\_socket < 0) {

perror("Accept failed");

continue;

}

printf("Connection accepted from %s:%d.\n", inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

client\_count++;

if ((childpid = fork()) == 0) {

close(server\_socket);

handle\_client(client\_socket, client\_count);

exit(0);

}

close(client\_socket);

}

close(server\_socket);

return 0;

}

Enhancement done in,

* Signal Handling: Added signal (SIGCHLD, SIG\_IGN); to prevent zombie processes by automatically reaping terminated child processes.
* Error Handling: Improved error messages using perror() for better diagnostics.
* Resource Management: Ensured sockets are properly closed in both parent and child processes.

**Echo Client code**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#define BUFFER\_SIZE 1024

#define PORT 8080

void communicate\_with\_server(int server\_socket) {

char buffer[BUFFER\_SIZE];

ssize\_t bytes\_read;

while (1) {

memset(buffer, 0, BUFFER\_SIZE);

printf("Msg to server: ");

fgets(buffer, BUFFER\_SIZE, stdin);

write(server\_socket, buffer, strlen(buffer));

memset(buffer, 0, BUFFER\_SIZE);

bytes\_read = read(server\_socket, buffer, sizeof(buffer));

if (bytes\_read <= 0) {

break;

}

printf("Msg from server: %s", buffer);

if (strncmp(buffer, "exit", 4) == 0) {

printf("Client Exit...\n");

break;

}

}

}

int main() {

int server\_socket;

struct sockaddr\_in server\_addr;

server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_socket < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

printf("Socket creation successful.\n");

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

if (connect(server\_socket, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) != 0) {

perror("Connection to the server failed");

close(server\_socket);

exit(EXIT\_FAILURE);

}

printf("Connected to the server.\n");

communicate\_with\_server(server\_socket);

close(server\_socket);

return 0;

}

Enhancement done in,

* Buffer Management: Used fgets() for reading input to avoid buffer overflow.
* Resource Management: Ensured the socket is closed properly upon exit.