Texas A&M University

ECEN 602 600: COMPUTER COMM & NET

Machine Problem-1

By: Chetan Sai Borra (UIN: 436000947)

Ganesh Vairavan Arumugam (UIN: 136004789)

Git code: https://github.com/Chetansai11/MP1_CCN602

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:

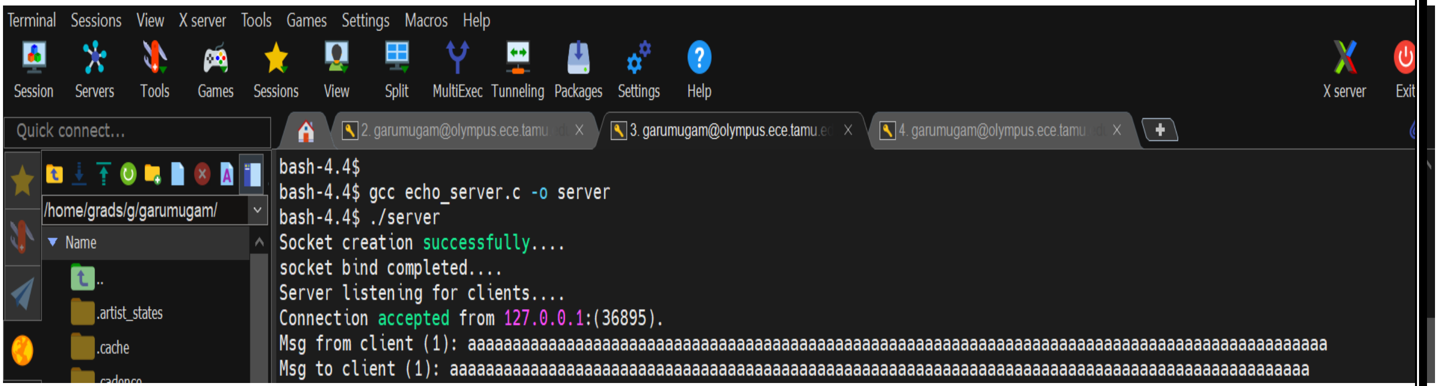
```
chetansai2003@DESKTOP-TPJHDFH:/mnt/c/Users/asus/Desktop/MS/CCN/MP1_CCN602/MP1$ gcc mp1sever.c -o server1
chetansai2003@DESKTOP-TPJHDFH:/mnt/c/Users/asus/Desktop/MS/CCN/MP1_CCN602/MP1$ ./server1
Socket creation successfully....
socket bind completed....
Server listening for clients....
Connection accepted from 127.0.0.1:(36895).
Msg from client (1): hi

Msg to client (1): hi

Msg from client (1): ok

Msg to client (1): ok

Msg from client (1): exit

Msg to client (1): exit
```

At Client side:

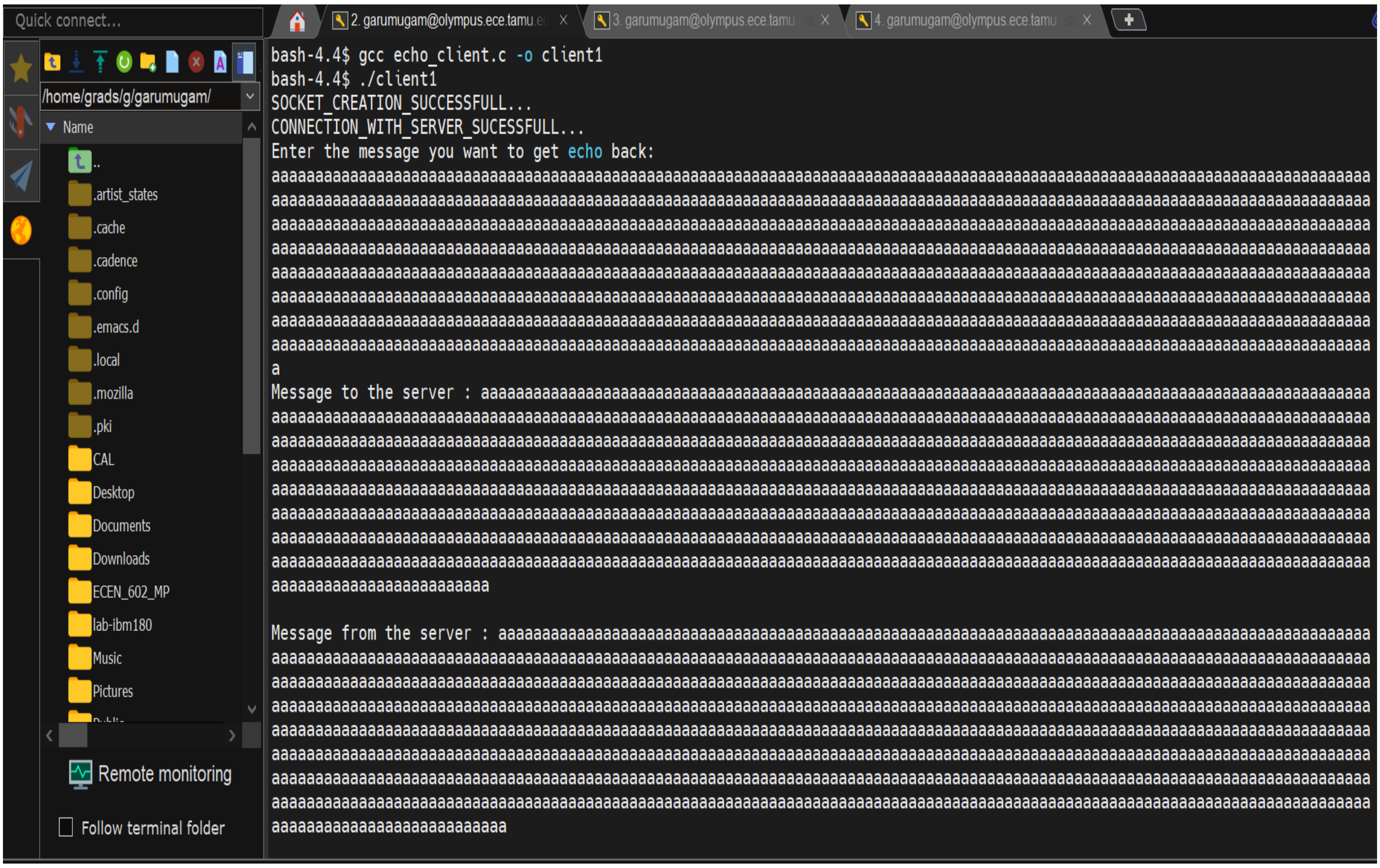
```
chetansai2003@DESKTOP-TPJHDFH:/mnt/c/Users/asus/Desktop/MS/CCN/MP1_CCN602/MP1$ gcc mp1client.c -o client1
chetansai2003@DESKTOP-TPJHDFH:/mnt/c/Users/asus/Desktop/MS/CCN/MP1_CCN602/MP1$ ./client1
Socket creation successfully....
connected to the server..
Msg to server: hi
Msg from Server : hi
Msg to server: ok
Msg from Server : ok
Msg to server: exit
Msg from Server : exit
```

2) **line of text the maximum line length without a newline**

At Server Side



At Client side



3) **line with nocharacters and EOF:**

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

At Server Side:

```
chetansai2003@DESKTOP-TPJHDFH:/mnt/c/Users/asus/Desktop/MS/CCN/MP1_CCN602/MP1$ gcc mp1sever.c -o server1
chetansai2003@DESKTOP-TPJHDFH:/mnt/c/Users/asus/Desktop/MS/CCN/MP1_CCN602/MP1$ ./server1
Socket creation successfully....
socket bind completed....
Server listening for clients....
Connection accepted from 127.0.0.1:(36895).
Msg from client (1): hi

Msg to client (1): hi

Msg from client (1):

Msg to client (1):

Msg from client (1):

Msg to client (1):

Msg from client (1): the above are lines with no char and eof

Msg to client (1): the above are lines with no char and eof
```

At Client Side:

```
chetansai2003@DESKTOP-TPJHDFH:/mnt/c/Users/asus/Desktop/MS/CCN/MP1_CCN602/MP1$ gcc mp1client.c -o client1
chetansai2003@DESKTOP-TPJHDFH:/mnt/c/Users/asus/Desktop/MS/CCN/MP1_CCN602/MP1$ ./client1
Socket creation successfully....
connected to the server..
Msg to server: hi
Msg from Server : hi
Msg to server:
Msg from Server :
Msg to server:
Msg from Server :
Msg to server: the above are lines with no char and eof
Msg from Server : the above are lines with no char and eof
```

4) client terminated after entering text:

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

At Server Side:

```
chetansai2003@DESKTOP-TPJHDFH:/mnt/c/Users/asus/Desktop/MS/CCN/MP1_CCN602/MP1$ ./server1
Socket creation successfully....
socket bind completed....
Server listening for clients....
Connection accepted from 127.0.0.1:(36895).
Msg from client (1): exit

Msg to client (1): exit

Client (1) Exited ...
```

At Client Side:

```
chetansai2003@DESKTOP-TPJHDFH:/mnt/c/Users/asus/Desktop/MS/CCN/MP1_CCN602/MP1$ gcc mp1client.c -o client1
chetansai2003@DESKTOP-TPJHDFH:/mnt/c/Users/asus/Desktop/MS/CCN/MP1_CCN602/MP1$ ./client1
Socket creation successfully....
connected to the server..
Msg to server: exit
Msg from Server : exit
Client Exit...
chetansai2003@DESKTOP-TPJHDFH:/mnt/c/Users/asus/Desktop/MS/CCN/MP1_CCN602/MP1$
```

5) Three clientsconnected to the server:

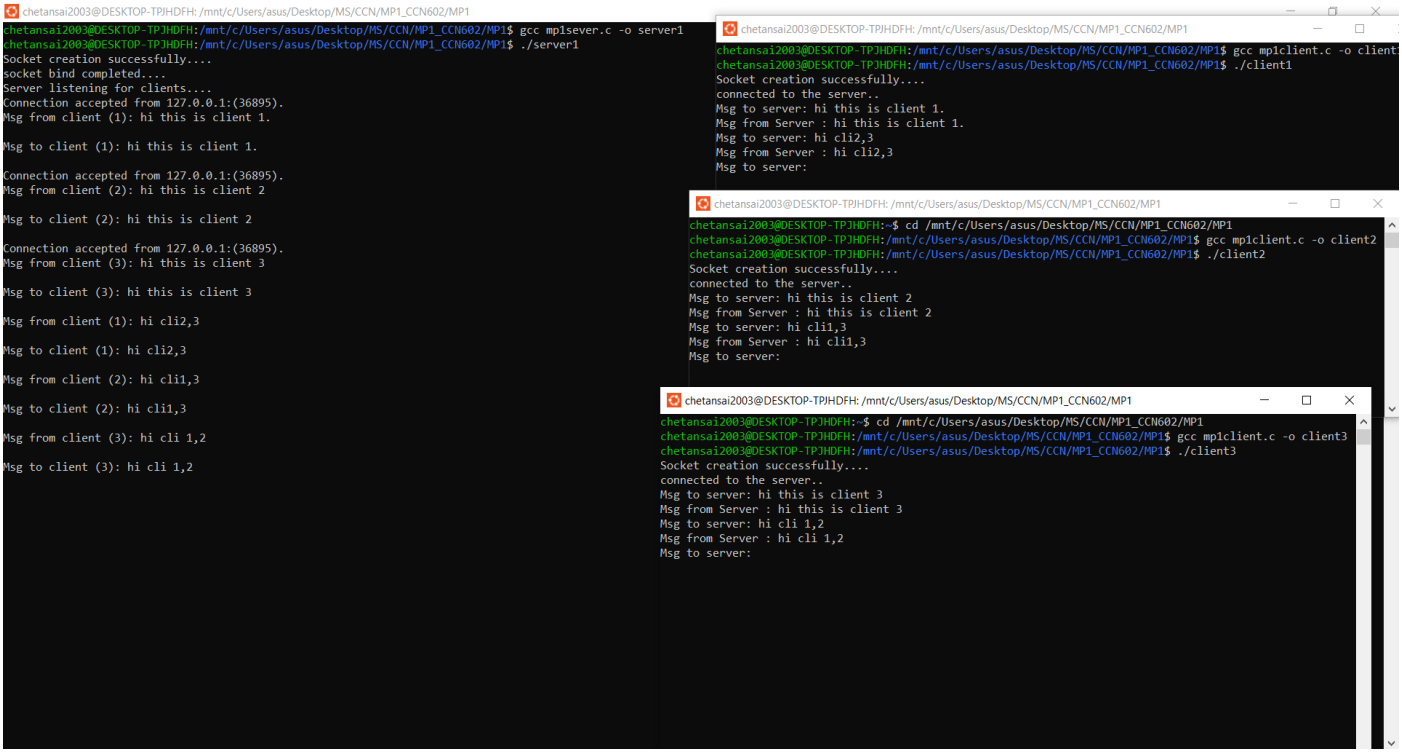
The server can connect to different clients simultaneously and echo the clients simultaneously.

At Server Side:

chetansai2003@DESKTOP-TPJHDFH: /mnt/c/Users/asus/Desktop/MS/CCN/MP1_CCN602/MP1

```
chetansai2003@DESKTOP-TPJHDFH:/mnt/c/Users/asus/Desktop/MS/CCN/MP1_CCN602/MP1$ gcc mp1sever.c -o server1
chetansai2003@DESKTOP-TPJHDFH:/mnt/c/Users/asus/Desktop/MS/CCN/MP1_CCN602/MP1$ ./server1
Socket creation successfully....
socket bind completed....
Server listening for clients....
Connection accepted from 127.0.0.1:(36895).
Msg from client (1): hi this is client 1.

Msg to client (1): hi this is client 1.

Connection accepted from 127.0.0.1:(36895).
Msg from client (2): hi this is client 2

Msg to client (2): hi this is client 2

Connection accepted from 127.0.0.1:(36895).
Msg from client (3): hi this is client 3

Msg to client (3): hi this is client 3

Msg from client (1): hi cli2,3

Msg to client (1): hi cli2,3

Msg from client (2): hi cli1,3

Msg to client (2): hi cli1,3

Msg from client (3): hi cli 1,2

Msg to client (3): hi cli 1,2
```

All the clients:



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;
}
```

2) 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.

Thank You!