Day – 7 LSP Assignment (Task – 1)

1. **Problem Statement: Socket Programming in C**

Design and implement a reliable and efficient network communication system using socket programming in C to enable data exchange between two or more processes running on different machines over a network.

Specific Requirements:Socket creation: Create appropriate socket descriptors for the desired communication protocol (TCP, UDP, etc.).

Address binding: Bind the created socket to a specific network address and port number for both client and server applications.

Connection establishment: Implement connection setup mechanisms (connect, accept) for TCP-based communication.

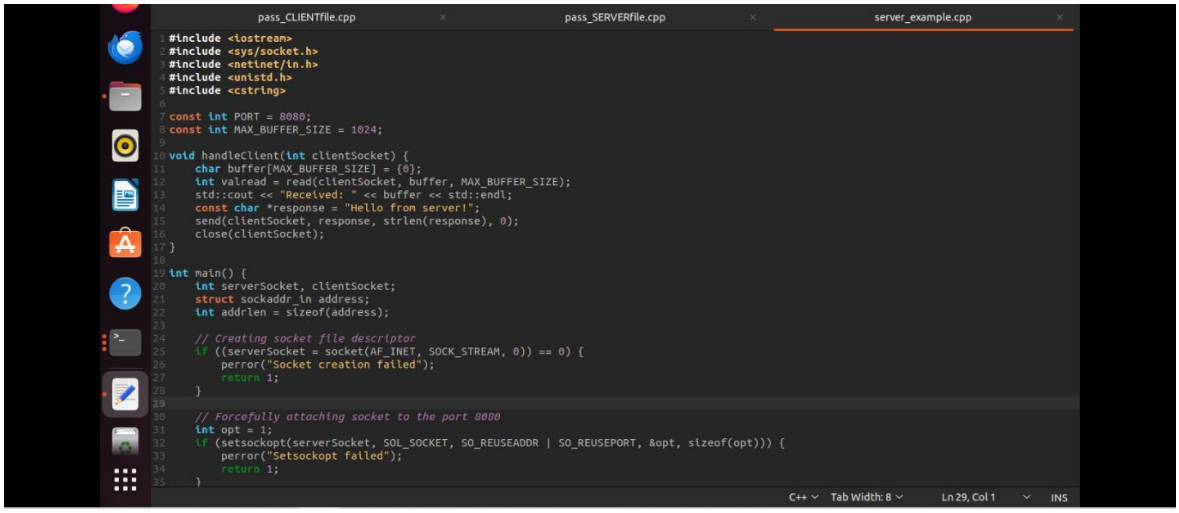
Data transfer: Develop functions for sending and receiving data over the established socket connection.

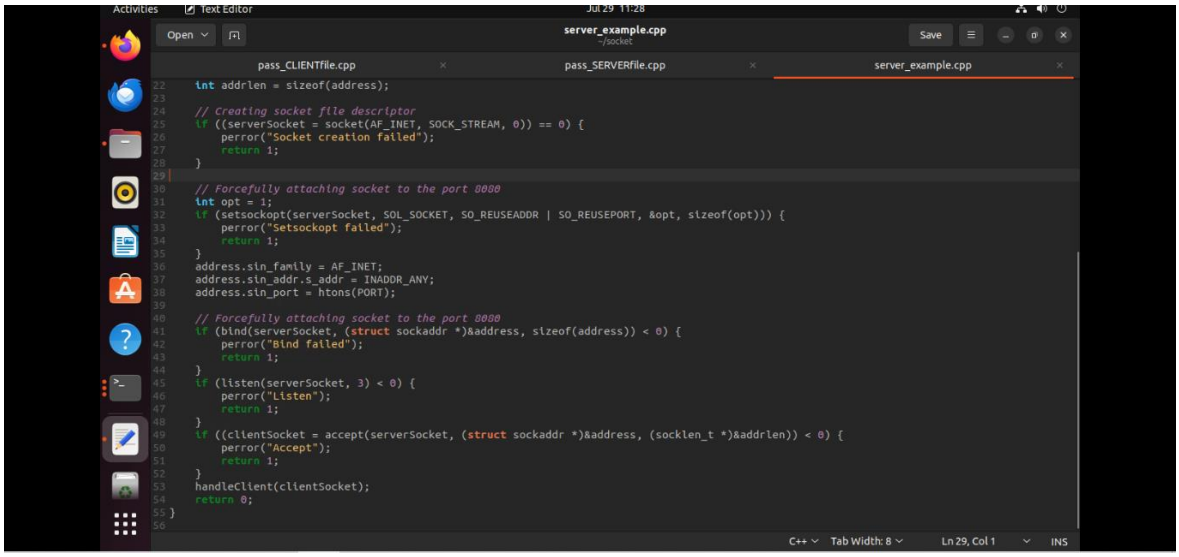
Error handling: Incorporate robust error handling mechanisms to address potential network issues and unexpected exceptions.

Concurrency: For server-side applications, consider handling multiple client connections concurrently using appropriate techniques (e.g., threading, forking).

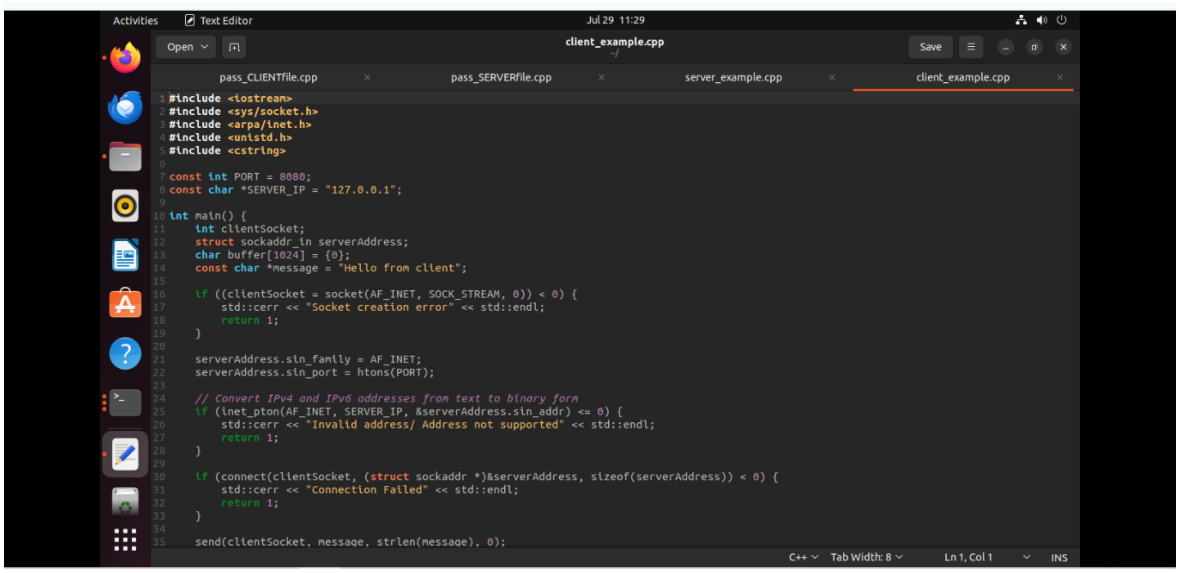
Security: Implement appropriate security measures to protect data integrity and confidentiality (e.g., encryption, authentication).

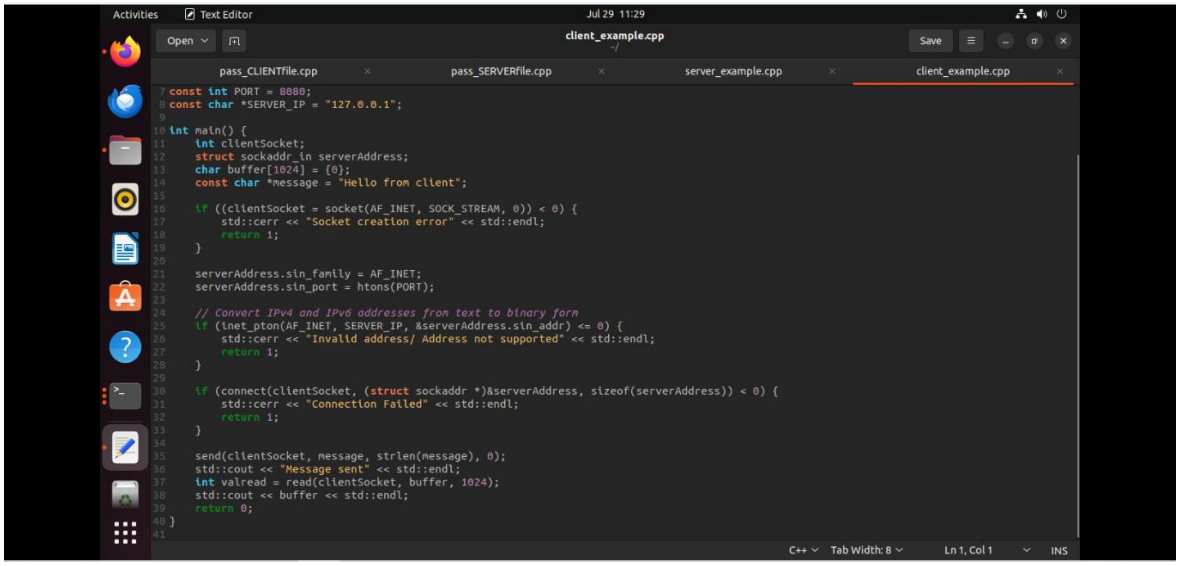
a. **server – side code**

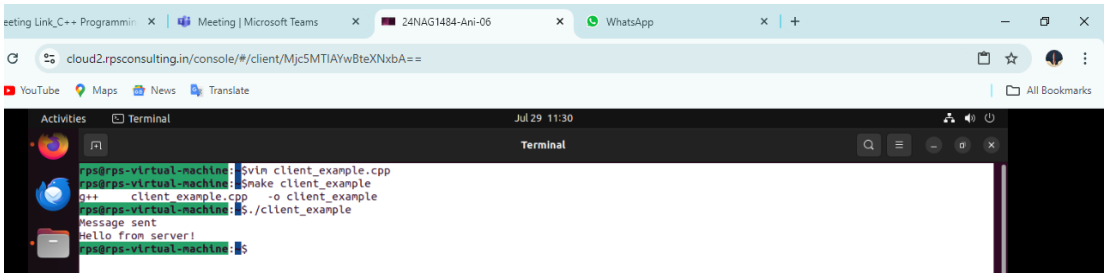




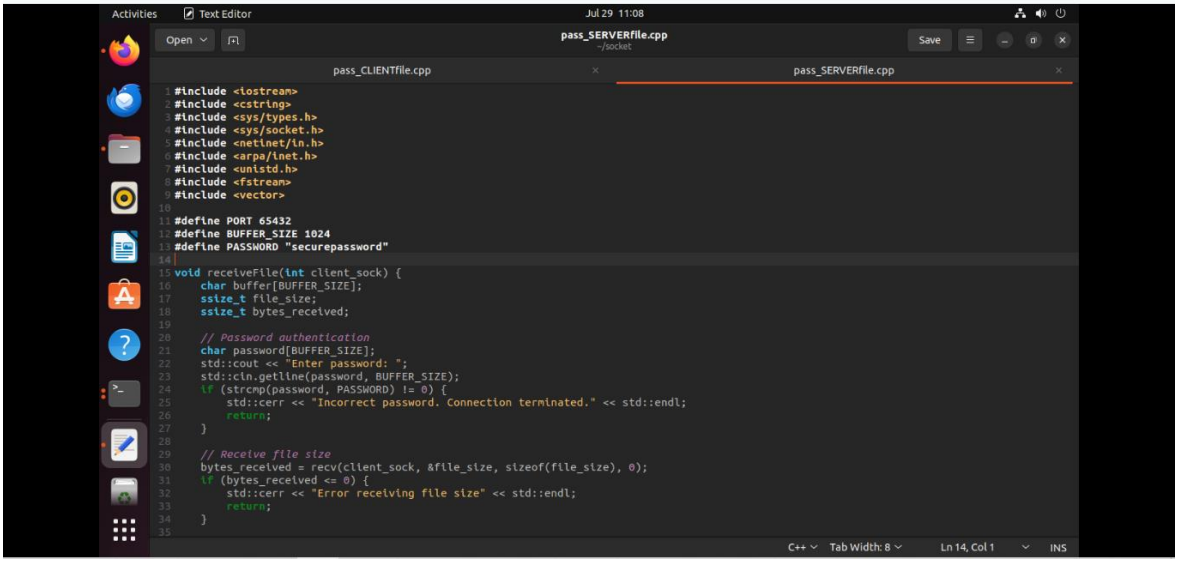
b. client – side code







2. **"Explain the role of password authentication and file transmission in your client-server application implementation."**
   a. **server - side code**



pass_CLIENTfile.cpp | pass_SERVERfile.cpp

```cpp
    std::ofstream file("received_SERVERfile.txt", std::ios::binary);
    if (!file.is_open()) {
        std::cerr << "Failed to open file to write" << std::endl;
        return;
    }

    ssize_t total_received = 0;
    while (total_received < file_size) {
        bytes_received = recv(client_sock, buffer, BUFFER_SIZE, 0);
        if (bytes_received <= 0) {
            std::cerr << "Error receiving file" << std::endl;
            return;
        }
        file.write(buffer, bytes_received);
        total_received += bytes_received;
    }

    std::cout << "File received from 'SERVER' successfully....." << std::endl;
    file.close();
}

void sendFile(int client_sock, const std::string& filename) {
    std::ifstream file(filename, std::ios::binary | std::ios::ate);
    if (!file.is_open()) {
        std::cerr << "Error opening file: " << filename << std::endl;
        return;
    }

    std::streamsize file_size = file.tellg();
    file.seekg(0, std::ios::beg);

    // Convert file size to network byte order
    file_size = htonl(file_size);
    send(client_sock, &file_size, sizeof(file_size), 0);
```

C++ Tab Width: 8 Ln 14, Col 1 INS

```cpp
    std::vector<char> buffer(BUFFER_SIZE);
    while (!file.eof()) {
        file.read(buffer.data(), buffer.size());
        ssize_t bytes_read = file.gcount();
        if (bytes_read <= 0) break;
        send(client_sock, buffer.data(), bytes_read, 0);
    }

    file.close();
    std::cout << "File sent from 'SERVER' successfully....." << std::endl;
}

int main() {
    int server_fd, client_sock;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);

    // 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 65432
    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);
```

C++ Tab Width: 8 Ln 14, Col 1 INS

```cpp

    // Forcefully attaching socket to the port 65432
    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 ((client_sock = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen)) < 0) {
        perror("accept");
        exit(EXIT_FAILURE);
    }

    // Handle client connection
    std::cout << "Client side connected" << std::endl;

    // Receive a command from the client
    char command[BUFFER_SIZE];
    ssize_t bytes_received = recv(client_sock, command, BUFFER_SIZE, 0);
    if (bytes_received <= 0) {
        std::cerr << "Error receiving command" << std::endl;
        close(client_sock);
        close(server_fd);
        return 1;
    }

    command[bytes_received] = '\0';

    // Check the received command
    if (strcmp(command, "get") == 0) {
        // Client requests to receive a file
        receiveFile(client_sock);
    } else if (strcmp(command, "send") == 0) {
        // Client requests to send a file
        std::string file_to_send = "pass_SERVERfile.txt"; // Replace with actual file name
        sendFile(client_sock, file_to_send);
    } else {
        std::cerr << "Invalid command received: " << command << std::endl;
    }

    close(client_sock);
    close(server_fd);
    return 0;
}

```

C++ Tab Width: 8 Ln 13, Col 1 INS

b. client – side code

```cpp
#include <iostream>
#include <cstring>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <fstream>
#include <vector>

#define PORT 65432
#define BUFFER_SIZE 1024
#define PASSWORD "securepassword"

void sendFile(int sock, const std::string& filename) {
    // Password authentication
    char password[BUFFER_SIZE];
    std::cout << "Enter password: ";
    std::cin.getline(password, BUFFER_SIZE);
    if (strcmp(password, PASSWORD) != 0) {
        std::cerr << "Incorrect password. Connection terminated." << std::endl;
        close(sock);
        return;
    }

    std::ifstream file(filename, std::ios::binary | std::ios::ate);
    if (!file.is_open()) {
        std::cerr << "Error opening file: " << filename << std::endl;
        return;
    }

    std::streamsize file_size = file.tellg();
    file.seekg(0, std::ios::beg);

    // Convert file size to network byte order
    file_size = htonl(file_size);
    send(sock, &file_size, sizeof(file_size), 0);

    std::vector<char> buffer(BUFFER_SIZE);
    while (!file.eof()) {
        file.read(buffer.data(), buffer.size());
        ssize_t bytes_read = file.gcount();
        if (bytes_read <= 0) break;
        send(sock, buffer.data(), bytes_read, 0);
    }

    file.close();
    std::cout << "File sent from 'CLIENT' successfully" << std::endl;
}

void receiveFile(int sock) {
    char buffer[BUFFER_SIZE];
    ssize_t file_size;
    ssize_t bytes_received;

    // Password authentication
    char password[BUFFER_SIZE];
    std::cout << "Enter password: ";
    std::cin.getline(password, BUFFER_SIZE);
    if (strcmp(password, PASSWORD) != 0) {
        std::cerr << "Incorrect password. Connection terminated." << std::endl;
        close(sock);
        return;
    }

    // Receive file size
    bytes_received = recv(sock, &file_size, sizeof(file_size), 0);
    if (bytes_received <= 0) {
        std::cerr << "Error receiving file size" << std::endl;
        return;
    }

    file_size = ntohl(file_size); // Convert from network byte order to host byte order

    std::ofstream file("received_SERVERfile.txt", std::ios::binary);
    if (!file.is_open()) {
        std::cerr << "Failed to open file to write" << std::endl;
        return;
    }

    ssize_t total_received = 0;
    while (total_received < file_size) {
        bytes_received = recv(sock, buffer, BUFFER_SIZE, 0);
        if (bytes_received <= 0) {
            std::cerr << "Error receiving file" << std::endl;
            return;
        }
        file.write(buffer, bytes_received);
        total_received += bytes_received;
    }

    std::cout << "File received from 'SERVER' successfully" << std::endl;
    file.close();
}

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[BUFFER_SIZE] = {0};
```

```cpp
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        std::cerr << "Socket creation error" << std::endl;
        return -1;
    }

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

    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        std::cerr << "Invalid address or Address not supported" << std::endl;
        close(sock);
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        std::cerr << "Connection failed" << std::endl;
        close(sock);
        return -1;
    }

    std::string file_to_send = "pass_CLIENTfile.txt"; // Replace with actual file name
    sendFile(sock, file_to_send);

    receiveFile(sock);

    close(sock);
    return 0;
}
```

C++ Tab Width: 8 Ln 1, Col 1 INS

3. Task - 3 : Implement code on message passing from client to server.

Activities Text Editor Jul 29 14:16

Open server_message.cpp ~/IPC_message Save

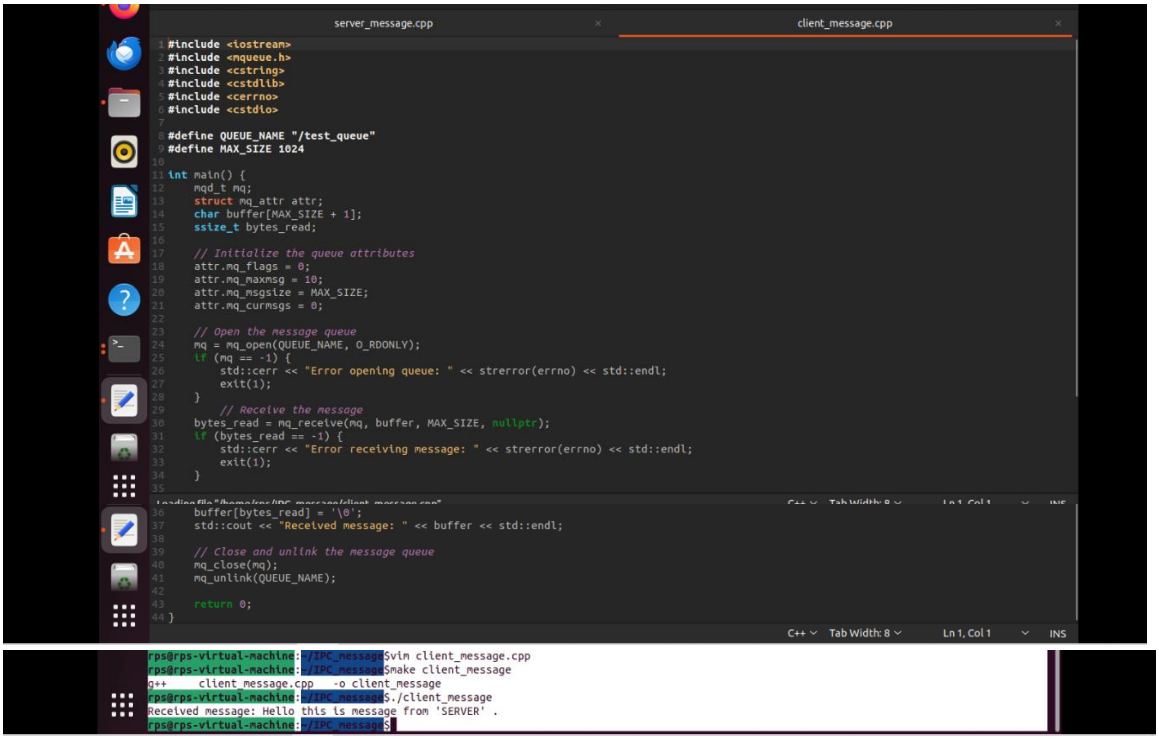
```cpp
#include <iostream>
#include <mqueue.h>
#include <cstring>
#include <cstdlib>
#include <cerrno>
#include <cstdio>

#define QUEUE_NAME "/test_queue"
#define MAX_SIZE 1024
#define MSG_STOP "exit"

int main() {
    mqd_t mq;
    struct mq_attr attr;
    char buffer[MAX_SIZE];

    // Initialize the queue attributes
    attr.mq_flags = 0;
    attr.mq_maxmsg = 10;
    attr.mq_msgsize = MAX_SIZE;
    attr.mq_curmsgs = 0;

    // Create the message queue
    mq = mq_open(QUEUE_NAME, O_CREAT | O_WRONLY, 0644, &attr);
    if (mq == -1) {
        std::cerr << "Error creating queue: " << strerror(errno) << std::endl;
        exit(1);
    }

    std::cout << "Enter a message: ";
    std::cin.getline(buffer, MAX_SIZE);

    // Send the message
    if (mq_send(mq, buffer, strlen(buffer) + 1, 0) == -1) {
        std::cerr << "Error sending message: " << strerror(errno) << std::endl;
        exit(1);
    }

    std::cout << "Message sent: " << buffer << std::endl;

    // Close the message queue
    mq_close(mq);

    return 0;
}
```

C++ Tab Width: 8 Ln 1, Col 1 INS

Bracket match not found C++ Tab Width: 8 Ln 26, Col 66 INS

```
rps@rps-virtual-machine:~/IPC_message$vim server_message.cpp
rps@rps-virtual-machine:~/IPC_message$make server_message
g++      server_message.cpp   -o server_message
rps@rps-virtual-machine:~/IPC_message$./server_message
Enter a message: Hello this is message from 'SERVER' .
Message sent: Hello this is message from 'SERVER' .
rps@rps-virtual-machine:~/IPC_message$
```

c. client_message



2. Concept on POSIX

4. Task – 4 : The functionality and robustness of POSIX message queue-based inter-process communication program in C++ .

a. server – code

```
        if (strcmp(buffer, MSG_STOP) == 0) break;

        // Receive a message
        ssize_t bytes_read = mq_receive(mq2, buffer, MAX_SIZE, nullptr);
        if (bytes_read == -1) {
            std::cerr << "Error receiving message: " << strerror(errno) << std::endl;
            exit(1);
        }
        buffer[bytes_read] = '\0';
        std::cout << "Process1 received: " << buffer << std::endl;
    }

    // Close and unlink the message queues
    mq_close(mq1);
    mq_close(mq2);
    mq_unlink(QUEUE_NAME1);
    mq_unlink(QUEUE_NAME2);

    return 0;
}
```

```
rps@rps-virtual-machine:~/IPC_message$./server_loop
Error opening queue: Bad address
rps@rps-virtual-machine:~/IPC_message$vim server_loop.cpp
rps@rps-virtual-machine:~/IPC_message$vim server_process.cpp
rps@rps-virtual-machine:~/IPC_message$make server_process
g++     server_process.cpp   -o server_process
rps@rps-virtual-machine:~/IPC_message$./server_process
Process1, enter a message: Hello How are you ?
Process1 received: I am fine.
Process1, enter a message: That's nice to hear.
Process1 received: and What about you ?
Process1, enter a message: I am fine too.
^Z
[20]+  Stopped                 ./server_process
rps@rps-virtual-machine:~/IPC_message$
```

b. client – side

```
#include <iostream>
#include <mqueue.h>
#include <cstring>
#include <cstdlib>
#include <cerrno>
#include <cstdio>

// Assuming MAX_SIZE and QUEUE_NAME1, QUEUE_NAME2 are defined in "common.h"
#include "common.h"

int main() {
    mqd_t mq1, mq2;
    struct mq_attr attr;
    char buffer[MAX_SIZE + 1]; // Increase buffer size by 1 for null termination

    // Initialize the queue attributes
    attr.mq_flags = 0;
    attr.mq_maxmsg = 10;
    attr.mq_msgsize = MAX_SIZE;
    attr.mq_curmsgs = 0;

    // Create the message queues
    mq1 = mq_open(QUEUE_NAME1, O_CREAT | O_RDONLY, 0644, &attr);
    mq2 = mq_open(QUEUE_NAME2, O_CREAT | O_WRONLY, 0644, &attr);
    if (mq1 == (mqd_t)-1 || mq2 == (mqd_t)-1) { // Cast -1 to mqd_t for comparison
        std::cerr << "Error creating queues: " << strerror(errno) << std::endl;
        exit(1);
    }

    while (true) {
        // Receive a message
        ssize_t bytes_read = mq_receive(mq1, buffer, MAX_SIZE, nullptr);
        if (bytes_read == -1) {
            std::cerr << "Error receiving message: " << strerror(errno) << std::endl;
            exit(1);
```

```
        }
        buffer[bytes_read] = '\0'; // Null-terminate received message
        std::cout << "Process2 received: " << buffer << std::endl;

        if (strcmp(buffer, MSG_STOP) == 0) break;

        // Send a message
        std::cout << "Process2, enter a message: ";
        std::cin.getline(buffer, MAX_SIZE); // Use getline to read a line of input
        if (mq_send(mq2, buffer, strlen(buffer) + 1, 0) == -1) {
            std::cerr << "Error sending message: " << strerror(errno) << std::endl;
            exit(1);
        }
    }

    // Close and unlink the message queues
    mq_close(mq1);
    mq_close(mq2);
    mq_unlink(QUEUE_NAME1);
    mq_unlink(QUEUE_NAME2);

    return 0;
}
```

```
rps@rps-virtual-machine:~/IPC_message$rm client_process.cpp
rps@rps-virtual-machine:~/IPC_message$vim client_process.cpp
rps@rps-virtual-machine:~/IPC_message$make client_process
g++     client_process.cpp   -o client_process
client_process.cpp:9:10: fatal error: common.h: No such file or directory
    9 | #include "common.h"
      |          ^~~~~~~~~~
compilation terminated.
make: *** [<builtin>: client_process] Error 1
rps@rps-virtual-machine:~/IPC_message$vim common.h
rps@rps-virtual-machine:~/IPC_message$make client_process
g++     client_process.cpp   -o client_process
rps@rps-virtual-machine:~/IPC_message$./client_process
Process2 received: Hello How are you ?
Process2, enter a message: I am fine.
Process2 received: That's nice to hear.
Process2, enter a message: and What about you ?
Process2 received: I am fine too.
Process2, enter a message: ^Z
[6]+  Stopped                 ./client_process
rps@rps-virtual-machine:~/IPC_message$
```

5. Task – 5 Use of semaphore for message passing.

a. Server –side

```
#include <iostream>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <cstring>
#include "common1.h"

int main() {
    // Open shared memory
    int shm_fd = shm_open(SHARED_MEMORY_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        std::cerr << "Error opening shared memory: " << strerror(errno) << std::endl;
        exit(1);
    }

    // Set size of shared memory
    if (ftruncate(shm_fd, sizeof(SharedMemory)) == -1) {
        std::cerr << "Error setting size of shared memory: " << strerror(errno) << std::endl;
        exit(1);
    }

    // Map shared memory
    SharedMemory *shared_memory = (SharedMemory *)mmap(nullptr, sizeof(SharedMemory), PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shared_memory == MAP_FAILED) {
        std::cerr << "Error mapping shared memory: " << strerror(errno) << std::endl;
        exit(1);
    }

    // Initialize semaphores
    sem_t *sem1 = sem_open(SEMAPHORE1_NAME, O_CREAT, 0666, 1);
    sem_t *sem2 = sem_open(SEMAPHORE2_NAME, O_CREAT, 0666, 0);
    if (sem1 == SEM_FAILED || sem2 == SEM_FAILED) {
        std::cerr << "Error opening semaphores: " << strerror(errno) << std::endl;
        exit(1);
    }

    // Initialize shared memory
    shared_memory->process1_turn = true;

    while (true) {
        // Wait for process1_turn to be true
        sem_wait(sem1);
        if (!shared_memory->process1_turn) {
            sem_post(sem1);
            continue;
        }

        // Write message to shared memory
        std::cout << "Process1, enter a message: ";
        std::cin.getline(shared_memory->buffer, MAX_SIZE);
        shared_memory->process1_turn = false;

        // Signal process 2
        sem_post(sem2);

        if (strcmp(shared_memory->buffer, "exit") == 0) break;
    }

    // Clean up
    munmap(shared_memory, sizeof(SharedMemory));
    close(shm_fd);
    sem_close(sem1);
    sem_close(sem2);
    sem_unlink(SEMAPHORE1_NAME);
    sem_unlink(SEMAPHORE2_NAME);
    shm_unlink(SHARED_MEMORY_NAME);

    return 0;
}
```

```
rps@rps-virtual-machine:~/IPC_message$vim server_semaphore.cpp
rps@rps-virtual-machine:~/IPC_message$vim server_semaphore.cpp
rps@rps-virtual-machine:~/IPC_message$g++ -o server_semaphore server_semaphore.cpp
rps@rps-virtual-machine:~/IPC_message$./server_semaphore
Process1, enter a message: Hello , Client !
Process1, enter a message: How is the process working client ?
Process1, enter a message: exit
rps@rps-virtual-machine:~/IPC_message$
```

b. client – side

```
#include <iostream>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <cstring>
#include "common1.h"

int main() {
    // Open shared memory
    int shm_fd = shm_open(SHARED_MEMORY_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        std::cerr << "Error opening shared memory: " << strerror(errno) << std::endl;
        exit(1);
    }

    // Set size of shared memory
    if (ftruncate(shm_fd, sizeof(SharedMemory)) == -1) {
        std::cerr << "Error setting size of shared memory: " << strerror(errno) << std::endl;
        exit(1);
    }

    // Map shared memory
    SharedMemory *shared_memory = (SharedMemory *)mmap(nullptr, sizeof(SharedMemory), PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shared_memory == MAP_FAILED) {
        std::cerr << "Error mapping shared memory: " << strerror(errno) << std::endl;
        exit(1);
    }

    // Initialize semaphores
    sem_t *sem1 = sem_open(SEMAPHORE1_NAME, O_CREAT, 0666, 1);
    sem_t *sem2 = sem_open(SEMAPHORE2_NAME, O_CREAT, 0666, 0);
    if (sem1 == SEM_FAILED || sem2 == SEM_FAILED) {
        std::cerr << "Error opening semaphores: " << strerror(errno) << std::endl;
        exit(1);
    }
```

server_semaphore.cpp | client_semaphore.cpp

```cpp
    while (true) {
        // Wait for process1_turn to be false
        sem_wait(sem2);
        if (shared_memory->process1_turn) {
            sem_post(sem2);
            continue;
        }

        // Read message from shared memory
        std::cout << "Process2 received: " << shared_memory->buffer << std::endl;

        if (strcmp(shared_memory->buffer, "exit") == 0) break;

        // Write response to shared memory
        std::cout << "Process2, enter a message: ";
        std::cin.getline(shared_memory->buffer, MAX_SIZE);
        shared_memory->process1_turn = true;

        // Signal process 1
        sem_post(sem1);

        if (strcmp(shared_memory->buffer, "exit") == 0) break;
    }

    // Clean up
    munmap(shared_memory, sizeof(SharedMemory));
    close(shm_fd);
    sem_close(sem1);
    sem_close(sem2);
    sem_unlink(SEMAPHORE1_NAME);
    sem_unlink(SEMAPHORE2_NAME);
    shm_unlink(SHARED_MEMORY_NAME);

    return 0;
}
```

C++ Tab Width: 8 Ln 1, Col 1 INS

```
rps@rps-virtual-machine:~/IPC_message$vim client_semaphore.cpp
rps@rps-virtual-machine:~/IPC_message$vim common1.h
rps@rps-virtual-machine:~/IPC_message$vim client_semaphore.cpp
rps@rps-virtual-machine:~/IPC_message$g++ -o client_semaphore client_semaphore.cpp
rps@rps-virtual-machine:~/IPC_message$./client_semaphore
Process2 received: Hello , Client !
Process2, enter a message: Hello, Server !
Process2 received: How is the process working client ?
Process2, enter a message: It's working fine.
Process2 received: exit
rps@rps-virtual-machine:~/IPC_message$
```

2. Concept on Pipe

a. Send – pipe

Activities | Text Editor | Jul 29 17:01

Open | pipe_send.cpp ~/IPC_message | Save

```cpp
#include <iostream>
#include <unistd.h>
#include <cstring>
#include <fcntl.h>
#include <sys/stat.h>
#include <cerrno>

#define PIPE1 "/tmp/pipe1"
#define PIPE2 "/tmp/pipe2"
#define MAX_SIZE 1024

void create_pipe(const char* pipe_name) {
    if (mkfifo(pipe_name, 0666) == -1) {
        if (errno != EEXIST) {
            std::cerr << "Error creating pipe " << pipe_name << ": " << strerror(errno) << std::endl;
            exit(1);
        }
    }
}

int main() {
    // Create the pipes
    create_pipe(PIPE1);
    create_pipe(PIPE2);

    int pipe1_fd, pipe2_fd;
    char buffer[MAX_SIZE];

    while (true) {
        // Write message to PIPE1
        std::cout << "Process1, enter a message: ";
        std::cin.getline(buffer, MAX_SIZE);
        pipe1_fd = open(PIPE1, O_WRONLY);
        if (pipe1_fd == -1) {
            std::cerr << "Error opening pipe1 for writing: " << strerror(errno) << std::endl;
            exit(1);
        }
```

C++ Tab Width: 8 Ln 1, Col 1 INS

```
        read(pipe1_fd, buffer, MAX_SIZE);
        std::cout << "Process2 received: " << buffer << std::endl;
        close(pipe1_fd);

        if (strcmp(buffer, "exit") == 0) break;

        // Write message to PIPE2
        std::cout << "Process2, enter a message: ";
        std::cin.getline(buffer, MAX_SIZE);
        pipe2_fd = open(PIPE2, O_WRONLY);
        if (pipe2_fd == -1) {
            std::cerr << "Error opening pipe2 for writing: " << strerror(errno) << std::endl;
            exit(1);
        }
        write(pipe2_fd, buffer, strlen(buffer) + 1);
        close(pipe2_fd);
    }

    std::cout << "Process2 exiting..." << std::endl;
    return 0;
}
```

```
Process1, enter a message: exit
rps@rps-virtual-machine:~/IPC_message$vim pipe_send.cpp
rps@rps-virtual-machine:~/IPC_message$vim pipe_send.cpp
rps@rps-virtual-machine:~/IPC_message$make pipe_send
g++      pipe_send.cpp   -o pipe_send
rps@rps-virtual-machine:~/IPC_message$./pipe_send
Process1, enter a message: Hello Client !
Process1 received: Hi, Server.
Process1, enter a message: exit
rps@rps-virtual-machine:~/IPC_message$
```

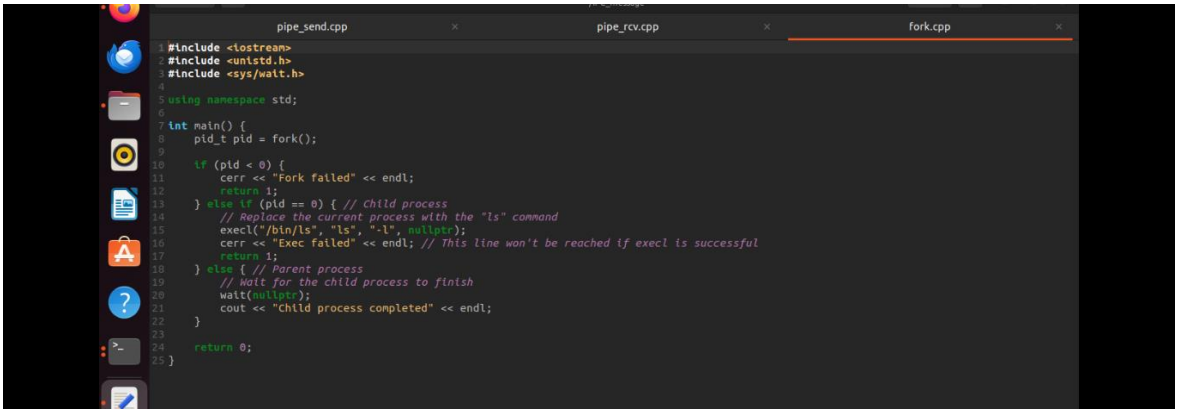
b. recv - pipe

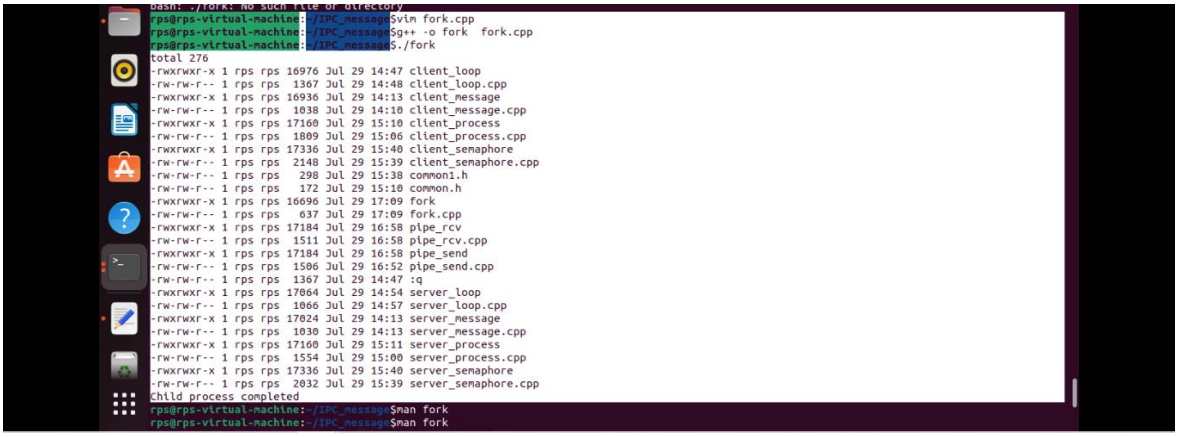
```
#include <iostream>
#include <unistd.h>
#include <cstring>
#include <fcntl.h>
#include <sys/stat.h>
#include <cerrno>

#define PIPE1 "/tmp/pipe1"
#define PIPE2 "/tmp/pipe2"
#define MAX_SIZE 1024

void create_pipe(const char* pipe_name) {
    if (mkfifo(pipe_name, 0666) == -1) {
        if (errno != EEXIST) {
            std::cerr << "Error creating pipe " << pipe_name << ": " << strerror(errno) << std::endl;
            exit(1);
        }
    }
}

int main() {
    // Create the pipes
    create_pipe(PIPE1);
    create_pipe(PIPE2);

    int pipe1_fd, pipe2_fd;
    char buffer[MAX_SIZE];

    while (true) {
        // Read message from PIPE1
        pipe1_fd = open(PIPE1, O_RDONLY);
        if (pipe1_fd == -1) {
            std::cerr << "Error opening pipe1 for reading: " << strerror(errno) << std::endl;
            exit(1);
        }
```

```
        write(pipe1_fd, buffer, strlen(buffer) + 1);
        close(pipe1_fd);

        if (strcmp(buffer, "exit") == 0) break;

        // Read message from PIPE2
        pipe2_fd = open(PIPE2, O_RDONLY);
        if (pipe2_fd == -1) {
            std::cerr << "Error opening pipe2 for reading: " << strerror(errno) << std::endl;
            exit(1);
        }
        read(pipe2_fd, buffer, MAX_SIZE);
        std::cout << "Process1 received: " << buffer << std::endl;
        close(pipe2_fd);

        if (strcmp(buffer, "exit") == 0) break;
    }

    return 0;
}
```

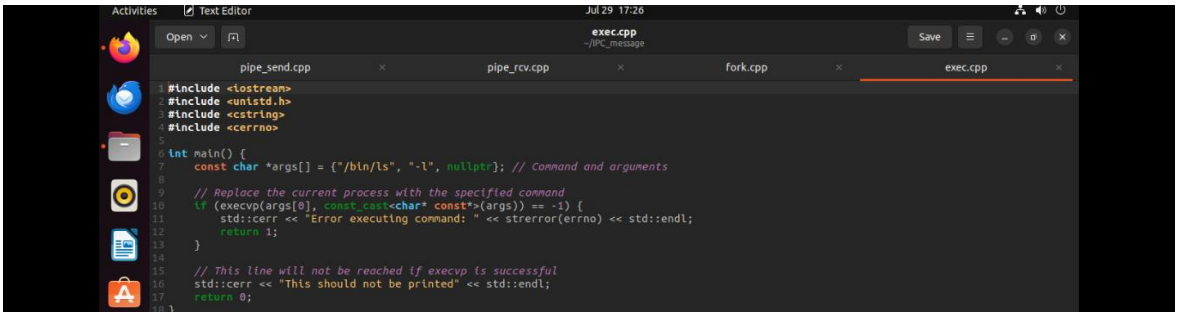
```
rps@rps-virtual-machine:~/IPC_message$vim pipe_rcv.cpp
rps@rps-virtual-machine:~/IPC_message$make pipe_rcv
g++      pipe_rcv.cpp   -o pipe_rcv
rps@rps-virtual-machine:~/IPC_message$./pipe_rcv
Process2 received: Hello Client !
Process2, enter a message: Hi, Server.
Process2 received: exit
Process2 exiting...
rps@rps-virtual-machine:~/IPC_message$
```

6. Task – 6 : Use of fork





7. Task – 7 : Use of exec



```
rps@rps-virtual-machine:~/IPC_message$rm exec.cpp
rps@rps-virtual-machine:~/IPC_message$vim exec.cpp
rps@rps-virtual-machine:~/IPC_message$g++ -o exec  exec.cpp
rps@rps-virtual-machine:~/IPC_message$./exec
total 300
-rwxrwxr-x 1 rps rps 16976 Jul 29 14:47 client_loop
-rw-rw-r-- 1 rps rps  1367 Jul 29 14:48 client_loop.cpp
-rwxrwxr-x 1 rps rps 16936 Jul 29 14:13 client_message
-rw-rw-r-- 1 rps rps  1038 Jul 29 14:10 client_message.cpp
-rwxrwxr-x 1 rps rps 17160 Jul 29 15:10 client_process
-rw-rw-r-- 1 rps rps  1809 Jul 29 15:06 client_process.cpp
-rwxrwxr-x 1 rps rps 17336 Jul 29 15:40 client_semaphore
-rw-rw-r-- 1 rps rps  2148 Jul 29 15:39 client_semaphore.cpp
-rw-rw-r-- 1 rps rps   298 Jul 29 15:38 common1.h
-rw-rw-r-- 1 rps rps   172 Jul 29 15:10 common.h
-rwxrwxr-x 1 rps rps 16720 Jul 29 17:15 exec
-rw-rw-r-- 1 rps rps   542 Jul 29 17:15 exec.cpp
-rwxrwxr-x 1 rps rps 16696 Jul 29 17:09 fork
-rw-rw-r-- 1 rps rps   637 Jul 29 17:11 fork.cpp
-rwxrwxr-x 1 rps rps 17184 Jul 29 16:58 pipe_rcv
-rw-rw-r-- 1 rps rps  1511 Jul 29 16:58 pipe_rcv.cpp
-rwxrwxr-x 1 rps rps 17184 Jul 29 16:58 pipe_send
-rw-rw-r-- 1 rps rps  1506 Jul 29 16:52 pipe_send.cpp
-rw-rw-r-- 1 rps rps  1367 Jul 29 14:47 :q
-rwxrwxr-x 1 rps rps 17064 Jul 29 14:54 server_loop
-rw-rw-r-- 1 rps rps  1066 Jul 29 14:57 server_loop.cpp
-rwxrwxr-x 1 rps rps 17024 Jul 29 14:13 server_message
-rw-rw-r-- 1 rps rps  1030 Jul 29 14:13 server_message.cpp
-rwxrwxr-x 1 rps rps 17160 Jul 29 15:11 server_process
-rw-rw-r-- 1 rps rps  1554 Jul 29 15:00 server_process.cpp
-rwxrwxr-x 1 rps rps 17336 Jul 29 15:40 server_semaphore
-rw-rw-r-- 1 rps rps  2032 Jul 29 15:39 server_semaphore.cpp
rps@rps-virtual-machine:~/IPC_message$man exec
rps@rps-virtual-machine:~/IPC_message$
```