

Day -11 LSP : Assignment

1. **Problem Statement:** Inter-Process Communication (IPC) using Pipes, Shared Memory, and Message Queues.

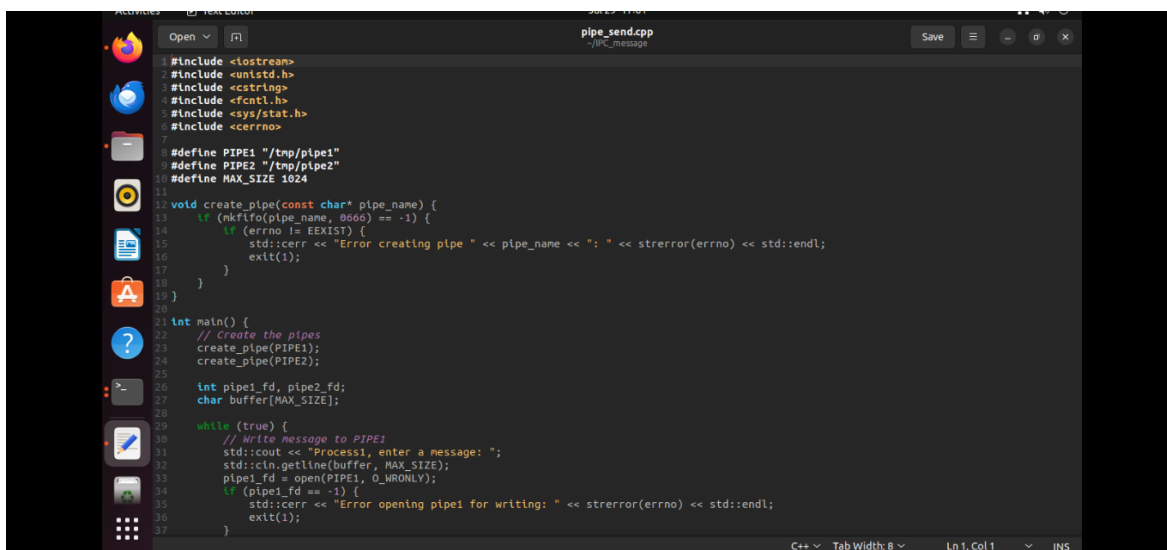
Design and implement efficient and reliable inter-process communication (IPC) mechanisms using pipes, shared memory, and message queues in C++ to facilitate data exchange and synchronization between multiple processes within a single system.

Specific Requirements:

- Pipe:** Create and manage unidirectional and bidirectional pipes for simple data transfer between related processes.
- Shared Memory:** Allocate and manage shared memory segments for efficient data sharing between multiple processes.
- Message Queues:** Create and utilize message queues for asynchronous communication and data exchange with message prioritization.
- Synchronization:** Implement appropriate synchronization mechanisms (e.g., semaphores, mutexes) to coordinate access to shared resources and prevent race conditions.
- Error Handling:** Incorporate robust error handling to manage potential IPC failures and resource leaks.

1. A. Pipe

a. pipe_send



```
#include <iostream>
#include <unistd.h>
#include <string>
#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);
        }
    }
}
```

```

36     read(pipe1_fd, buffer, MAX_SIZE);
37     std::cout << "Process2 received: " << buffer << std::endl;
38     close(pipe1_fd);
39
40     if (strcmp(buffer, "exit") == 0) break;
41
42     // Write message to PIPE2
43     std::cout << "Process2, enter a message: ";
44     std::cin.getline(buffer, MAX_SIZE);
45     pipe2_fd = open(PIPE2, O_WRONLY);
46     if (pipe2_fd == -1) {
47         std::cerr << "Error opening pipe2 for writing: " << strerror(errno) << std::endl;
48         exit(1);
49     }
50     write(pipe2_fd, buffer, strlen(buffer) + 1);
51     close(pipe2_fd);
52 }
53
54 std::cout << "Process2 exiting..." << std::endl;
55 return 0;
56 }
57

```

```

Process1, enter a message: exit
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. rcv_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);
        }

        read(pipe1_fd, buffer, MAX_SIZE);
        std::cout << "Process1 received: " << buffer << std::endl;
        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 << "Process2 received: " << buffer << std::endl;
        close(pipe2_fd);

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

    return 0;
}

```

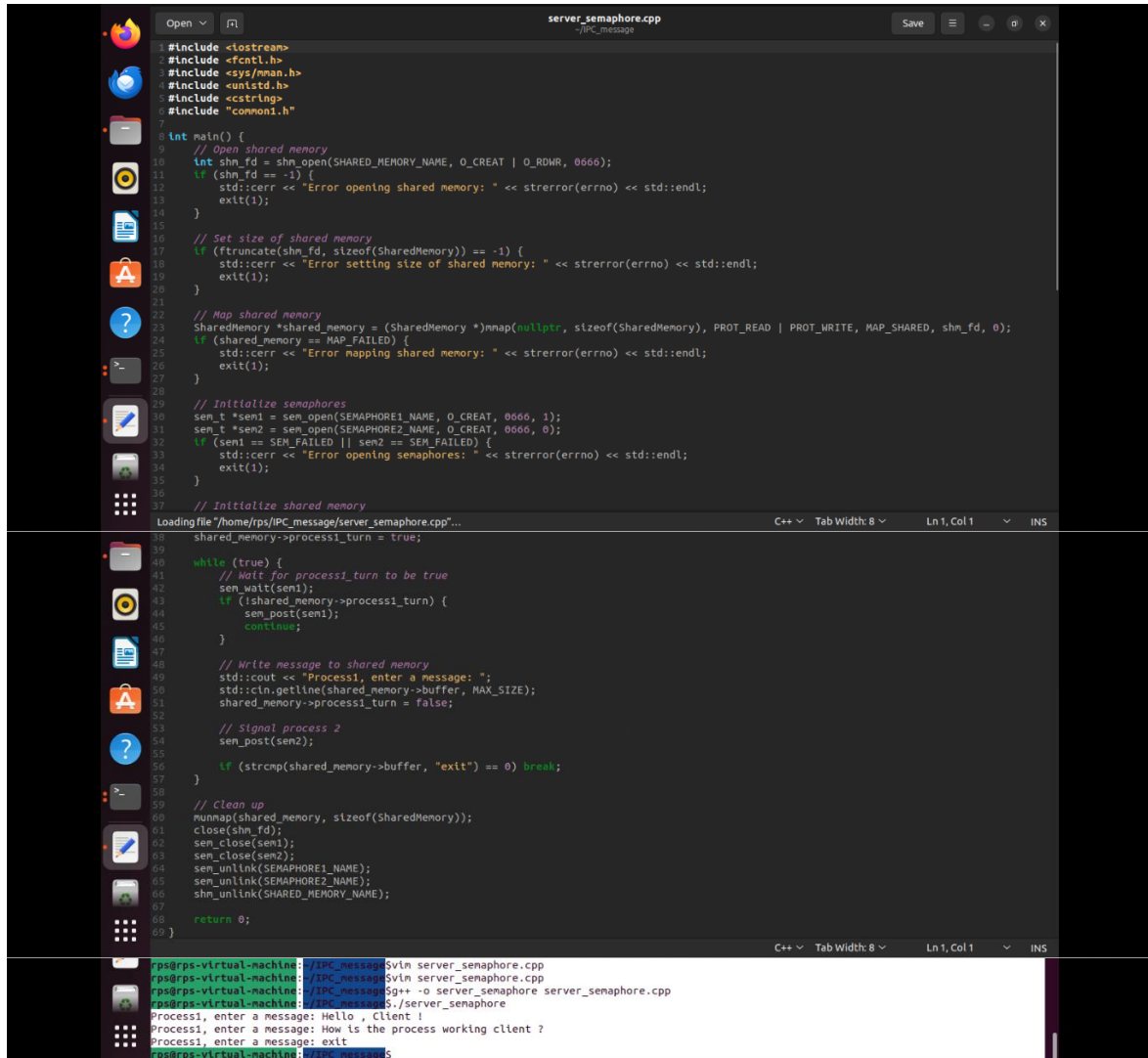
```

Terminal
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 $

```

2. A. Shared Memory

a. Server – side



```
#include <iostream>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <string>
#include "common.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;
}
```

Loading file "/home/rps/IPC_message/server_semaphore.cpp"...

shared_memory->process1_turn = true;

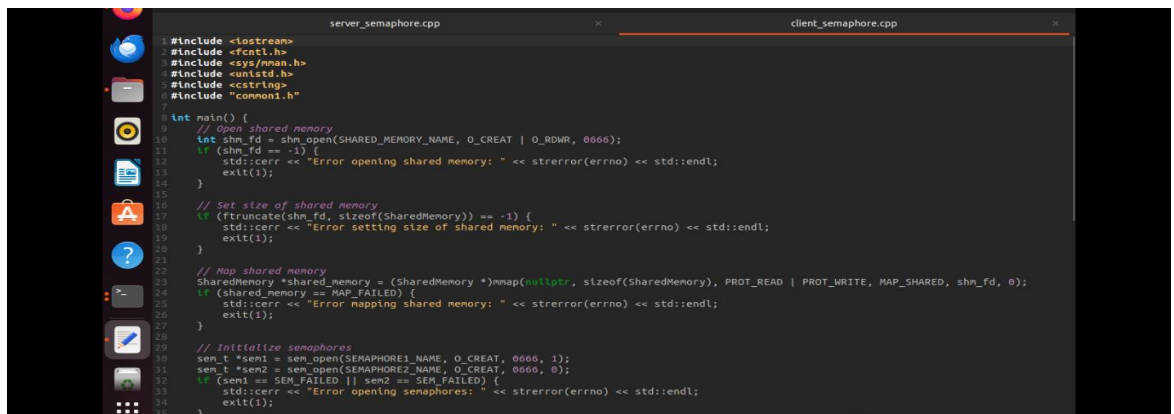
Process1, enter a message: Hello , Client !

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 <string>
#include "common.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;
}
```

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\$

```

server_semaphore.cpp
37 while (true) {
38     // wait for process1_turn to be false
39     sem_wait(sem2);
40     if (shared_memory->process1_turn) {
41         sem_post(sem2);
42         continue;
43     }
44     // Read message from shared memory
45     std::cout << "Process2 received: " << shared_memory->buffer << std::endl;
46     if (strcmp(shared_memory->buffer, "exit") == 0) break;
47     // Write response to shared memory
48     std::cout << "Process2, enter a message: ";
49     std::cin.getline(shared_memory->buffer, MAX_SIZE);
50     shared_memory->process1_turn = true;
51     // Signal process 1
52     sem_post(sem1);
53     if (strcmp(shared_memory->buffer, "exit") == 0) break;
54 }
55 // Clean up
56 munmap(shared_memory, sizeof(SharedMemory));
57 close(shm_fd);
58 sem_close(sem1);
59 sem_close(sem2);
60 sem_unlink(SEMAPHORE1_NAME);
61 sem_unlink(SEMAPHORE2_NAME);
62 shm_unlink(SharedMemory_NAME);
63 return 0;
64 }

```

```

rps@rps-virtual-machine:~/IPC_message$ g++ client_semaphore.cpp
rps@rps-virtual-machine:~/IPC_message$ g++ common1.h
rps@rps-virtual-machine:~/IPC_message$ g++ 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$

```

3. A. Message queue

a. server – message

```

server_message.cpp
1 #include <iostream>
2 #include <queue>
3 #include <string>
4 #include <cstdlib>
5 #include <cerrno>
6 #include <stdio.h>
7
8 #define QUEUE_NAME "/test_queue"
9 #define MAX_SIZE 1024
10 #define MSG_STOP "exit"
11
12 int main() {
13     mqd_t mq;
14     struct mq_attr attr;
15     char buffer[MAX_SIZE];
16
17     // Initialize the queue attributes
18     attr.mq_flags = 0;
19     attr.mq_maxmsg = 10;
20     attr.mq_msgsize = MAX_SIZE;
21     attr.mq_curmsgs = 0;
22
23     // Create the message queue
24     mq = mq_open(QUEUE_NAME, O_CREAT | O_WRONLY, 0644, &attr);
25     if (mq == -1) {
26         std::cerr << "Error creating queue: " << strerror(errno) << std::endl;
27         exit(1);
28     }
29     std::cout << "Enter a message: ";
30     std::cin.getline(buffer, MAX_SIZE);
31
32     // Send the message
33     if (mq_send(mq, buffer, strlen(buffer) + 1, 0) == -1) {
34         std::cerr << "Error sending message: " << strerror(errno) << std::endl;
35         exit(1);
36     }
37 }
38
39 std::cout << "Message sent: " << buffer << std::endl;
40
41 // Close the message queue
42 mq_close(mq);
43 return 0;
44 }

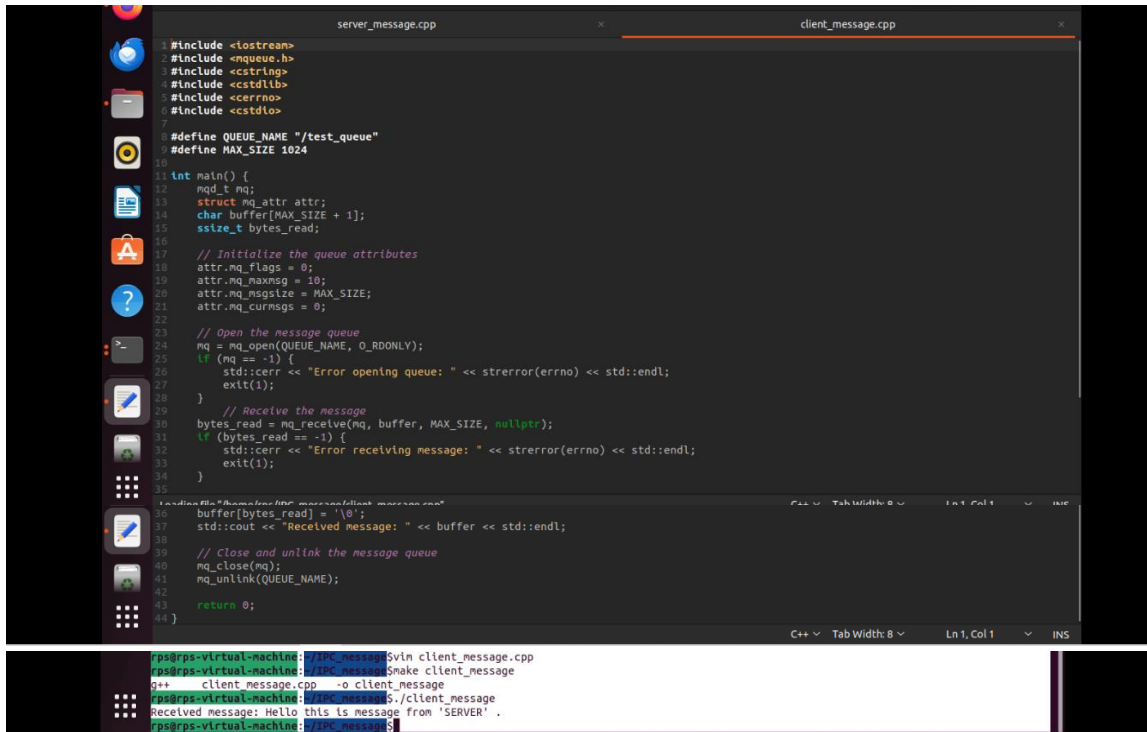
```

```

rps@rps-virtual-machine:~/IPC_message$ g++ server_message.cpp
rps@rps-virtual-machine:~/IPC_message$ g++ server_message.cpp
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$

```

b. client message



```
server_message.cpp
#include <iostream>
#include <queue.h>
#include <string>
#include <stdlib.h>
#include <errno>
#include <stdio.h>

#define QUEUE_NAME "/test_queue"
#define MAX_SIZE 1024

int main() {
    mqd_t mq;
    struct mq_attr attr;
    char buffer[MAX_SIZE + 1];
    ssize_t bytes_read;

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

    // Open the message queue
    mq = mq_open(QUEUE_NAME, O_RDONLY);
    if (mq == -1) {
        std::cerr << "Error opening queue: " << strerror(errno) << std::endl;
        exit(1);
    }

    // Receive the message
    bytes_read = mq_receive(mq, buffer, MAX_SIZE, nullptr);
    if (bytes_read == -1) {
        std::cerr << "Error receiving message: " << strerror(errno) << std::endl;
        exit(1);
    }

    // Print the received message
    std::cout << "Received message: " << buffer << std::endl;

    // Close and unlink the message queue
    mq_close(mq);
    mq_unlink(QUEUE_NAME);

    return 0;
}

client_message.cpp
#include <iostream>
#include <queue.h>
#include <string>
#include <stdlib.h>
#include <errno>
#include <stdio.h>

#define QUEUE_NAME "/test_queue"
#define MAX_SIZE 1024

int main() {
    mqd_t mq;
    struct mq_attr attr;
    char buffer[MAX_SIZE + 1];
    ssize_t bytes_read;

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

    // Open the message queue
    mq = mq_open(QUEUE_NAME, O_WRONLY);
    if (mq == -1) {
        std::cerr << "Error opening queue: " << strerror(errno) << std::endl;
        exit(1);
    }

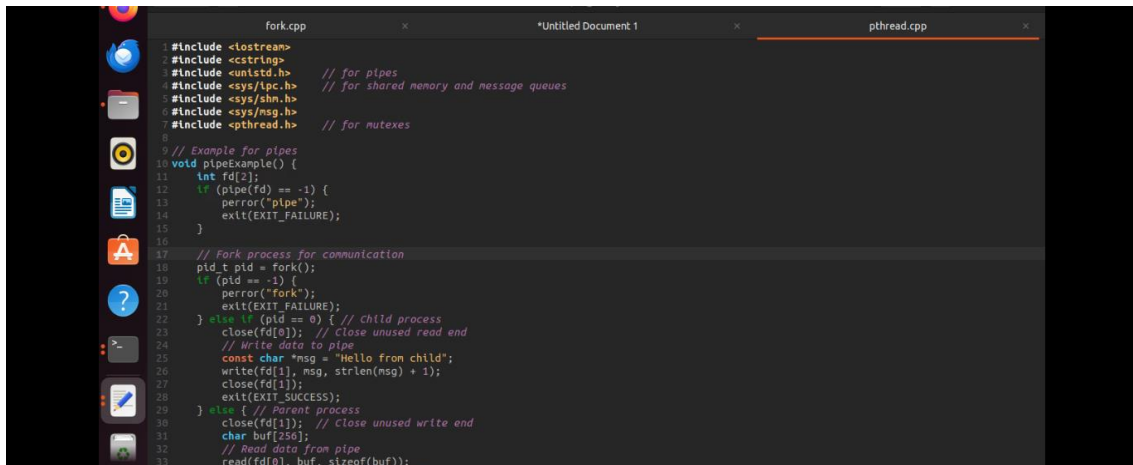
    // Send the message
    const char *msg = "Hello this is message from 'SERVER' .";
    if (mq_send(mq, msg, strlen(msg) + 1, 0) == -1) {
        std::cerr << "Error sending message: " << strerror(errno) << std::endl;
        exit(1);
    }

    // Close the message queue
    mq_close(mq);

    return 0;
}

Terminal Output:
rps@rps-virtual-machine:~/C++_Messages$ g++ client_message.cpp
rps@rps-virtual-machine:~/C++_Messages$ make client_message
g++ client_message.cpp -o client_message
rps@rps-virtual-machine:~/C++_Messages$ ./client_message
Received message: Hello this is message from 'SERVER' .
rps@rps-virtual-machine:~/C++_Messages$
```

The entire assignment code can be coded as follows,



```
fork.cpp
#include <iostream>
#include <string>
#include <unistd.h> // for pipes
#include <sys/types.h> // for shared memory and message queues
#include <sys/shm.h>
#include <sys/msg.h>
#include <pthread.h> // for mutexes

// Example for pipes
void pipeExample() {
    int fd[2];
    if (pipe(fd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    // Fork process for communication
    pid_t pid = fork();
    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (pid == 0) { // child process
        close(fd[0]); // Close unused read end
        // Write data to pipe
        const char *msg = "Hello from child";
        write(fd[1], msg, strlen(msg) + 1);
        close(fd[1]);
        exit(EXIT_SUCCESS);
    } else { // Parent process
        close(fd[1]); // Close unused write end
        char buf[256];
        // Read data from pipe
        read(fd[0], buf, sizeof(buf));
    }
}
```

```

34     std::cout << "Received from child: " << buf << std::endl;
35     close(fd[0]);
36 }
37 }
38
39 // Example for shared memory
40 void sharedMemoryExample() {
41     key_t key = ftok("/tmp", 'A');
42     int shmid = shmget(key, sizeof(int), 0666|IPC_CREAT);
43     if (shmid == -1) {
44         perror("shmget");
45         exit(EXIT_FAILURE);
46     }
47     int *data = (int*) shmat(shmid, NULL, 0);
48     if (data == (int*) -1) {
49         perror("shmat");
50         exit(EXIT_FAILURE);
51     }
52     *data = 42;
53     std::cout << "Shared memory value: " << *data << std::endl;
54     shmdt(data);
55     shmctl(shmid, IPC_RMID, NULL);
56 }
57
58 // Example for message queues
59 struct MyMsgbuf {
60     long mtype; // message type, must be > 0
61     char mtext[256]; // message data
62 };
63
64 void messageQueueExample() {
65     key_t key = ftok("/tmp", 'B');
66     int msqid = msgget(key, 0666|IPC_CREAT);
67     if (msqid == -1) {
68         perror("msgget");
69         exit(EXIT_FAILURE);
70     }
71
72     struct MyMsgbuf buf;
73     buf.mtype = 1; // message type, can be used for prioritization
74
75     // Sending message
76     const char *msg_send = "Hello from sender";
77     strncpy(buf.mtext, msg_send, sizeof(buf.mtext));
78     if (msgsnd(msqid, &buf, sizeof(buf.mtext), 0) == -1) {
79         perror("msgsnd");
80         exit(EXIT_FAILURE);
81     }
82     std::cout << "Message sent: " << msg_send << std::endl;
83
84     // Receiving message
85     if (msgrcv(msqid, &buf, sizeof(buf.mtext), 1, 0) == -1) {
86         perror("msgrcv");
87         exit(EXIT_FAILURE);
88     }
89     std::cout << "Message received: " << buf.mtext << std::endl;
90
91     // Cleanup
92     msgctl(msqid, IPC_RMID, NULL);
93 }
94
95 int main() {
96     std::cout << "=== Pipe Example ===" << std::endl;
97     pipeExample();
98     std::cout << "\n=== Shared Memory Example ===" << std::endl;
99     sharedMemoryExample();
100
101     std::cout << "\n=== Message Queue Example ===" << std::endl;
102     messageQueueExample();
103
104     return 0;
105 }
106
107

```

Execution :

```

rps@rps-virtual-machine: ~/IPC_message$ vim pthread.cpp
rps@rps-virtual-machine: ~/IPC_message$ make pthread
g++ pthread.cpp -o pthread
rps@rps-virtual-machine: ~/IPC_message$ ./pthread
=== Pipe Example ===
Received from child: Hello from child

=== Shared Memory Example ===
Shared memory value: 42

=== Message Queue Example ===
Message sent: Hello from sender
Message received: Hello from sender
rps@rps-virtual-machine: ~/IPC_message$

```

- Task:** Create a program that replicates the functionality of the standard cp command, but without using any standard library functions related to file I/O. Instead, you must employ system calls directly to perform file operations.

Requirements:

System calls: Utilize system calls like open, close, read, and write to interact with files.

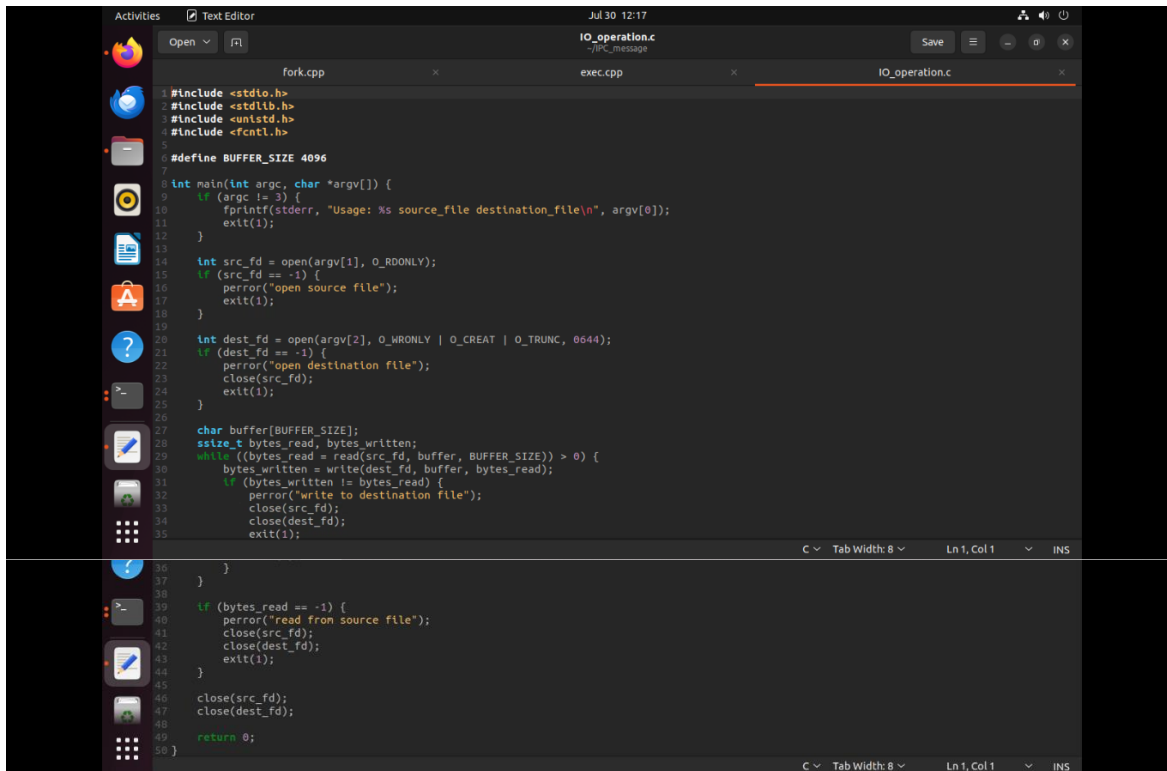
Error handling: Implement robust error handling for potential issues such as file not found, permission denied, disk full, etc.

Efficiency: Optimize the copying process for performance, considering buffer sizes and read/write operations.

Metadata: Preserve file permissions, timestamps, and other relevant metadata during the copy process.

User interface: Provide a simple command-line interface with options for source and destination file paths.

a. code



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <fcntl.h>

#define BUFFER_SIZE 4096

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s source_file destination_file\n", argv[0]);
        exit(1);
    }

    int src_fd = open(argv[1], O_RDONLY);
    if (src_fd == -1) {
        perror("open source file");
        exit(1);
    }

    int dest_fd = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (dest_fd == -1) {
        perror("open destination file");
        close(src_fd);
        exit(1);
    }

    char buffer[BUFFER_SIZE];
    ssize_t bytes_read, bytes_written;
    while ((bytes_read = read(src_fd, buffer, BUFFER_SIZE)) > 0) {
        bytes_written = write(dest_fd, buffer, bytes_read);
        if (bytes_written != bytes_read) {
            perror("write to destination file");
            close(src_fd);
            close(dest_fd);
            exit(1);
        }
    }

    if (bytes_read == -1) {
        perror("read from source file");
        close(src_fd);
        close(dest_fd);
        exit(1);
    }

    close(src_fd);
    close(dest_fd);
    return 0;
}
```

Execution :



```
rps@rps-virtual-machine: ~/IPC_message$ g++ IO_operation.cpp -o IO_operation
rps@rps-virtual-machine: ~/IPC_message$ ./IO_operation
Usage: ./IO_operation source_file destination_file
rps@rps-virtual-machine: ~/IPC_message$ g++ IO_operation.cpp -o IO_operation
rps@rps-virtual-machine: ~/IPC_message$ ./IO_operation
Usage: ./IO_operation source_file destination_file
rps@rps-virtual-machine: ~/IPC_message$
```