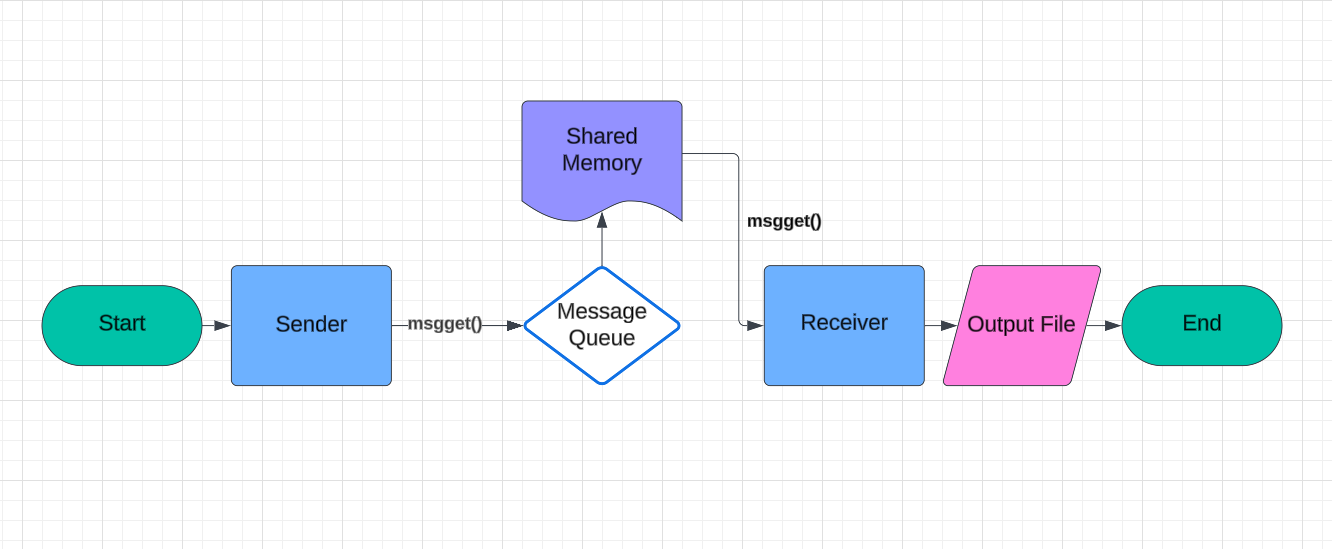
## **Design of Sender and Receiver**

## The goal of the sender and receiver programs is to facilitate file transfer between two processes using message queues and shared memory. This document will explain the sender and receiver programs' architecture, key components, and logic.

### **2. System Overview**

### The sender and receiver programs perform the utilization of various functions to communicate via a message queue and shared memory. The sender reads a file and transfers its contents to the receiver via shared memory, with the message queue coordinating the data transfer. This process is simplified and visualized in the Flow Diagram pictured below.

**Flow Diagram**:



### **3. Sender Design**

#### **3.1. Initialization**

**Message Queue Creation**: The sender program creates a message queue utilizing the msgget() function with a unique key derived from the ftok() function.

**Shared Memory Creation**: The sender creates a shared memory segment using the shmget() function and attaches it to its corresponding address space using shmat() function.

#### **3.2. File Reading and Transfer**

**File Handling**: The sender opens the input file and reads it in chunks.

**Data Writing**: The sender writes each chunk of data to the shared memory segment.

**Message Sending**: After writing a chunk to shared memory, the sender sends a message to the receiver using the msgsnd() function, indicating that new data has become available.

#### **3.3. Cleanup**

**Resource Deallocation**: Once the file transfer is complete, the sender detaches from shared memory using shmdt() and deallocates it using shmctl(). The message queue is also removed using msgctl().

### **4. Receiver Design**

#### **4.1. Initialization**

**Message Queue Access**: The receiver accesses the existing message queue using msgget() with the same key used by the sender.

**Shared Memory Access**: The receiver attaches to the shared memory segment using shmat().

#### **4.2. Data Receiving and Writing**

**Message Receiving**: The receiver waits for messages from the sender using msgrcv().

**File Writing**: Upon receiving a message, the receiver reads the corresponding data from shared memory and writes it to the output file.

#### **4.3. Cleanup**

**Resource Deallocation**: After the file transfer is complete, the receiver detaches from shared memory using shmdt() and deallocates it using shmctl(). The message queue is also removed using msgctl().

### **5. Signal Handling**

**Signal Handling Implementation**: Both sender and receiver programs implement a signal handler for SIGINT using signal(). This handler calls cleanUp() to ensure that shared memory and message queues are properly deallocated on interruption.

### **6. Error Handling**

**Error Checking**: All system calls, including ftok(), shmget(), shmat(), msgget(), msgrcv(), and msgsnd(), are checked for errors. If any call fails, perror() is used to print the error message, and the program exits.

### **7. Conclusion**

**Summary**: This document outlines the design of the sender and receiver programs, highlighting the key components and their interactions. The Flow Diagram demonstrates a simple representation of how the sender and receiver interact with one another and the shared memory in the program. From information on initialization to cleanup processes used once the program has completed its primary task. Proper usage of message queues and shared memory ensures efficient and reliable file transfer between processes. With added details on Signal Handling processes and the implementation of Error Handling which demonstrates actions taken if errors occur.