**IPC: Message Queues**

**Subject - Unix Operating System**

**Name – Hemant Sharma**

**PRN – 22610001 Class – TYIT**

**Assignment No – 7c**

**Title-** Write a program and two programs so that they can communicate privately to individually via a message queue.

**Objectives-**

1. To learn about IPC through message queue.
2. Use of system call and IPC mechanism to write effective application programs.

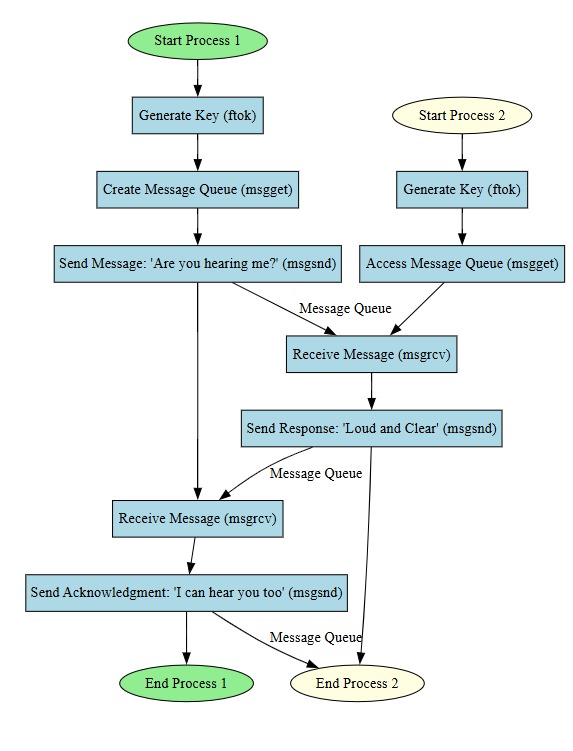
**Theory:**

A message queue is a linked list of messages stored within the kernel and identified by a message queue identifier. A new queue is created or an existing queue opened by msgget().

New messages are added to the end of a queue by msgsnd(). Every message has a positive long integer type field, a non-negative length, and the actual data bytes (corresponding to the length), all of which are specified to msgsnd() when the message is added to a queue. Messages are fetched from a queue by msgrcv(). We don’t have to fetch the messages in a first-in, first-out order. Instead, we can fetch messages based on their type field.

All processes can exchange information through access to a common system message queue. The sending process places a message (via some (OS) message-passing module) onto a queue which can be read by another process. Each message is given an identification or type so that processes can select the appropriate message. Process must share a common key in order to gain access to the queue in the first place.

**Flowchart:**

****

**Program:  
Process1:**

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <string.h>

struct msg\_buffer {

    long msg\_type;

    char msg\_text[100];

};

int main() {

    key\_t key;

    int msgid;

    struct msg\_buffer message;

    // Generate unique key

    key = ftok("progfile", 65);

    // Create message queue

    msgid = msgget(key, 0666 | IPC\_CREAT);

    // Sending initial message

    message.msg\_type = 1;

    strcpy(message.msg\_text, "Are you hearing me?");

    msgsnd(msgid, &message, sizeof(message.msg\_text), 0);

    printf("Process 1: Sent -> %s\n", message.msg\_text);

    // Receiving reply from Process 2

    msgrcv(msgid, &message, sizeof(message.msg\_text), 2, 0);

    printf("Process 1: Received -> %s\n", message.msg\_text);

    // Sending final acknowledgment

    message.msg\_type = 3;

    strcpy(message.msg\_text, "I can hear you too");

    msgsnd(msgid, &message, sizeof(message.msg\_text), 0);

    printf("Process 1: Sent -> %s\n", message.msg\_text);

    return 0;

}

**Process2:**

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <string.h>

struct msg\_buffer {

    long msg\_type;

    char msg\_text[100];

};

int main() {

    key\_t key;

    int msgid;

    struct msg\_buffer message;

    // Generate unique key

    key = ftok("progfile", 65);

    // Access message queue

    msgid = msgget(key, 0666 | IPC\_CREAT);

    // Receiving message from Process 1

    msgrcv(msgid, &message, sizeof(message.msg\_text), 1, 0);

    printf("Process 2: Received -> %s\n", message.msg\_text);

    // Sending response

    message.msg\_type = 2;

    strcpy(message.msg\_text, "Loud and Clear");

    msgsnd(msgid, &message, sizeof(message.msg\_text), 0);

    printf("Process 2: Sent -> %s\n", message.msg\_text);

    return 0;

}

**Output:**

****

**Conclusion:**

Use of message queue functions like msgget, msgsend, and msgrecv to implement message passing mechanism between server and client studied. Various clients communicated independently with server