**IPC: Message Queues**

**Subject - Unix Operating System**

**Name – Hemant Sharma**

**PRN – 22610001 Class – TYIT**

**Assignment No – 7a**

**Title-** Write a program to perform IPC using message and send did u get this? and then reply.

**Objectives-**

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

**Theory:**

Two (or more) processes can exchange information via access to a common system message queue. The sending process places via some (OS) message-passing module a message onto a queue which can be read by another process

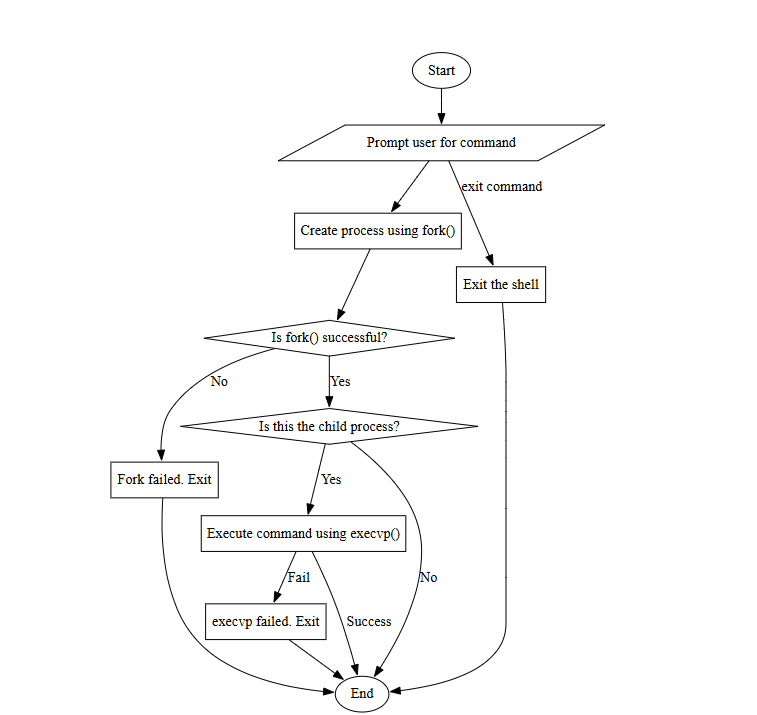
Each message is given an identification or type so that processes can select the appropriate message. Process must share a common key in order to gain access to the queue in the first place.

Basic Message Passing IPC messaging lets processes send and receive messages, and queue messages for processing in an arbitrary order. Unlike the file byte-stream data flow of pipes, each IPC message has an explicit length. Messages can be assigned a specific type. Because of this, a server process can direct message traffic between clients on its queue by using the client process PID as the message type. For single-message transactions, multiple server processes can work in parallel on transactions sent to a shared message queue.

When a message is sent, its text is copied to the message queue. The msgsnd() and msgrcv() functions can be performed as either blocking or non-blocking operations. Non-blocking operations allow for asynchronous message transfer -- the process is not suspended as a result of sending or receiving a message. In blocking or synchronous message passing the sending process cannot continue until the message has been transferred or has even been acknowledged by a receiver. IPC signal and other mechanisms can be employed to implement such transfer. A blocked message operation remains suspended until one of the following three conditions occurs:

1. The call succeeds.
2. The process receives a signal.
3. The queue is removed

**Flowchart:**

****

**Program:**

**Sender:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <mqueue.h>

#include <fcntl.h>

#include <unistd.h>

#define QUEUE\_NAME\_SEND "/sender\_to\_receiver"

#define QUEUE\_NAME\_REPLY "/receiver\_to\_sender"

#define MAX\_MSG\_SIZE 256

int main() {

    mqd\_t mq\_send\_queue, mq\_reply\_queue;

    char \*msg = "Did you get this?";

    char buffer[MAX\_MSG\_SIZE];

    // Set the message queue attributes

    struct mq\_attr attr;

    attr.mq\_flags = 0;

    attr.mq\_maxmsg = 10;  // Max number of messages

    attr.mq\_msgsize = MAX\_MSG\_SIZE;  // Max message size

    attr.mq\_curmsgs = 0;

    // Create the sending message queue (for sender to receiver)

    mq\_send\_queue = mq\_open(QUEUE\_NAME\_SEND, O\_CREAT | O\_WRONLY, 0666, &attr);

    if (mq\_send\_queue == -1) {

        perror("Sender: Message queue creation/open failed");

        exit(1);

    }

    // Create the reply message queue (for receiver to sender)

    mq\_reply\_queue = mq\_open(QUEUE\_NAME\_REPLY, O\_CREAT | O\_RDONLY, 0666, &attr);

    if (mq\_reply\_queue == -1) {

        perror("Sender: Reply queue creation/open failed");

        exit(1);

    }

    // Send the message

    if (mq\_send(mq\_send\_queue, msg, strlen(msg) + 1, 0) == -1) {

        perror("Sender: Message send failed");

        mq\_close(mq\_send\_queue);

        mq\_close(mq\_reply\_queue);

        exit(1);

    }

    printf("Sender: Sent message: %s\n", msg);

    // Wait for the reply from the receiver

    ssize\_t bytes\_read = mq\_receive(mq\_reply\_queue, buffer, MAX\_MSG\_SIZE, NULL);

    if (bytes\_read == -1) {

        perror("Sender: Reply receive failed");

        mq\_close(mq\_send\_queue);

        mq\_close(mq\_reply\_queue);

        exit(1);

    }

    printf("Sender: Received reply: %s\n", buffer);

    // Close the message queues

    mq\_close(mq\_send\_queue);

    mq\_close(mq\_reply\_queue);

    return 0;

}

**Receiver;**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <mqueue.h>

#include <fcntl.h>

#include <unistd.h>

#define QUEUE\_NAME\_SEND "/sender\_to\_receiver"

#define QUEUE\_NAME\_REPLY "/receiver\_to\_sender"

#define MAX\_MSG\_SIZE 256

int main() {

    mqd\_t mq\_send\_queue, mq\_receive\_queue;

    char buffer[MAX\_MSG\_SIZE];

    char \*reply = "Yes, I got it!";

    // Set the message queue attributes

    struct mq\_attr attr;

    attr.mq\_flags = 0;

    attr.mq\_maxmsg = 10;  // Max number of messages

    attr.mq\_msgsize = MAX\_MSG\_SIZE;  // Max message size

    attr.mq\_curmsgs = 0;

    // Open the sending message queue (for sender to receiver)

    mq\_receive\_queue = mq\_open(QUEUE\_NAME\_SEND, O\_RDONLY);

    if (mq\_receive\_queue == -1) {

        perror("Receiver: Message queue open failed");

        exit(1);

    }

    // Open the reply message queue (for receiver to sender)

    mq\_send\_queue = mq\_open(QUEUE\_NAME\_REPLY, O\_CREAT | O\_WRONLY, 0666, &attr);

    if (mq\_send\_queue == -1) {

        perror("Receiver: Reply queue creation/open failed");

        exit(1);

    }

    // Receive the message

    ssize\_t bytes\_read = mq\_receive(mq\_receive\_queue, buffer, MAX\_MSG\_SIZE, NULL);

    if (bytes\_read == -1) {

        perror("Receiver: Message receive failed");

        mq\_close(mq\_send\_queue);

        mq\_close(mq\_receive\_queue);

        exit(1);

    }

    printf("Receiver: Received message: %s\n", buffer);

    // Send the reply

    if (mq\_send(mq\_send\_queue, reply, strlen(reply) + 1, 0) == -1) {

        perror("Receiver: Reply send failed");

        mq\_close(mq\_send\_queue);

        mq\_close(mq\_receive\_queue);

        exit(1);

    }

    printf("Receiver: Sent reply: %s\n", reply);

    // Close the message queues

    mq\_close(mq\_send\_queue);

    mq\_close(mq\_receive\_queue);

    return 0;

}

**Output:**

**Conclusion:**

Use of message queue functions like msgget, msgsend, and msgrecv to implement message passing mechanism between server and client studied and implemented it to introduce concept of chatting.