**IPC: Message Queues**

**Subject - Unix Operating System**

**Name – Hemant Sharma**

**PRN – 22610001 Class – TYIT**

**Assignment No – 7b**

**Title-** Write 2 programs that will both send and messages and construct the following dialog between them

**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

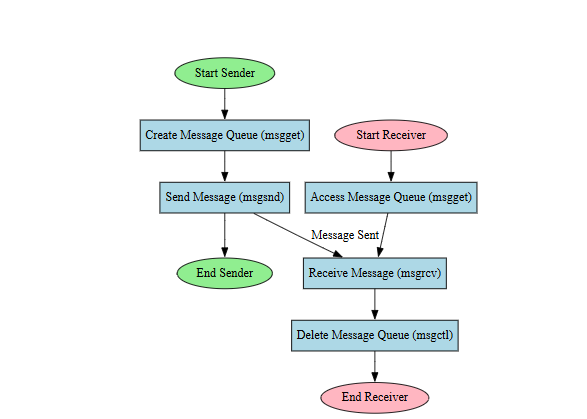
Each message is given an identification or type so that processes can select the appropriate message. Process must share a common key in order to gain access to the queue in the first place.

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 <unistd.h>

#include <arpa/inet.h>

#define SERVER\_IP "127.0.0.1" // Localhost

#define SERVER\_PORT 12345     // Port to connect to

void sender\_program() {

    int sock;

    struct sockaddr\_in server\_addr;

    char \*messages[] = {

        "Hello, Receiver!",

        "How are you doing?",

        "Do you have any updates for me?",

        "Okay, I'll wait for your response."

    };

    char buffer[1024];

    // Create socket

    sock = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sock < 0) {

        perror("Socket creation failed");

        exit(1);

    }

    // Set up server address structure

    server\_addr.sin\_family = AF\_INET;

    server\_addr.sin\_port = htons(SERVER\_PORT);

    server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

    // Connect to the server

    if (connect(sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

        perror("Connection failed");

        close(sock);

        exit(1);

    }

    // Send messages and receive responses

    for (int i = 0; i < 4; i++) {

        send(sock, messages[i], strlen(messages[i]), 0);

        printf("Sent: %s\n", messages[i]);

        // Receive response from server

        int n = recv(sock, buffer, sizeof(buffer) - 1, 0);

        if (n < 0) {

            perror("Receive failed");

            close(sock);

            exit(1);

        }

        buffer[n] = '\0';  // Null-terminate the response

        printf("Receiver's response: %s\n", buffer);

    }

    // Close the socket

    close(sock);

}

int main() {

    sender\_program();

    return 0;

}

**Receiver:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define SERVER\_PORT 12345     // Port to listen on

void receiver\_program() {

    int server\_sock, client\_sock;

    struct sockaddr\_in server\_addr, client\_addr;

    socklen\_t client\_len;

    char buffer[1024];

    // Create socket

    server\_sock = socket(AF\_INET, SOCK\_STREAM, 0);

    if (server\_sock < 0) {

        perror("Socket creation failed");

        exit(1);

    }

    // Set up server address structure

    server\_addr.sin\_family = AF\_INET;

    server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

    server\_addr.sin\_port = htons(SERVER\_PORT);

    // Bind the socket

    if (bind(server\_sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

        perror("Binding failed");

        close(server\_sock);

        exit(1);

    }

    // Listen for incoming connections

    if (listen(server\_sock, 5) < 0) {

        perror("Listen failed");

        close(server\_sock);

        exit(1);

    }

    printf("Waiting for a connection...\n");

    // Accept a connection from the sender

    client\_len = sizeof(client\_addr);

    client\_sock = accept(server\_sock, (struct sockaddr \*)&client\_addr, &client\_len);

    if (client\_sock < 0) {

        perror("Accept failed");

        close(server\_sock);

        exit(1);

    }

    // Receive messages and send responses

    while (1) {

        int n = recv(client\_sock, buffer, sizeof(buffer) - 1, 0);

        if (n < 0) {

            perror("Receive failed");

            close(client\_sock);

            close(server\_sock);

            exit(1);

        }

        buffer[n] = '\0';  // Null-terminate the received message

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

        // Respond to the sender

        char \*response = "Message received!";

        send(client\_sock, response, strlen(response), 0);

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

    }

    // Close the sockets

    close(client\_sock);

    close(server\_sock);

}

int main() {

    receiver\_program();

    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.