Mayank Lakhotia
2022PMN4226

Practical – 5

Aim: -

Implement TWO-WAY Inter-process communication using Message Queues. Consider TWO independent processes for communication. This communication must continue till a specific key is pressed or a STOP message is sent by any one of the processes.

Theory: -

A message queue is a linked list of messages stored within the kernel and identified by a message queue identifier. System calls used for message queues are as follows:

- A new queue is created or an existing queue opened by **msgget ()**.
- New messages are added to the end of a queue by **msgsnd ()**.
- Messages are fetched from a queue by **msgrcv ().**
- **msgctl ()**: It performs various operations on a queue. Generally, it is use to destroy message queue.
- **ftok ()**: is use to generate a unique key.

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.

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.

Mayank Lakhotia
2022PMN4226

Source Code: - C++ Language

client.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAX_TEXT 512

struct my_msg_st {
    long int my_msg_type;
    char some_text[BUFSIZ];
};

int main()
{
    int running = 1;
    int msgid, msgid2;
    struct my_msg_st some_data, send_data;
    long int msg_to_receive = 0;
    char buffer[BUFSIZ];

    send_data.my_msg_type = 1;
```

```
    msgid = msgget((key_t)2345, 0666 | IPC_CREAT);
    msgid2 = msgget((key_t)1234, 0666 | IPC_CREAT);

if (msgid == -1 || msgid2 == -1) {
    fprintf(stderr, "msgget failed with error: %d\n", errno);
    exit(EXIT_FAILURE);
}

while(running) {
    printf("Enter a message from client : ");
    fgets(buffer, BUFSIZ, stdin);
    send_data.my_msg_type = 1;
    strcpy(send_data.some_text, buffer);

    if (msgsnd(msgid2, (void *)&send_data, MAX_TEXT, 0) == -1) {
        fprintf(stderr, "msgsnd failed\n");
        exit(EXIT_FAILURE);
}
if (strncmp(buffer, "stop", 4) == 0) {
    running = 0;
} else {
    printf("\nWaiting for server message... \n");
    if (msgrcv(msgid, (void *)&some_data, BUFSIZ, msg_to_receive, 0) == -1) {
        fprintf(stderr, "msgrcv failed with error %d\n", errno);
        exit(EXIT_FAILURE);
}
printf("\nMessage received from server  %s", some_data.some_text);
if (strncmp(some_data.some_text, "stop", 4) == 0) {
    running = 0;
```

```
}
}
}
if (msgctl(msgid, IPC_RMID, 0) == -1) {
    fprintf(stderr, "msgctl(IPC_RMID) failed\n");
    exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```

server.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAX_TEXT 512

struct my_msg_st {
    long int my_msg_type;
    char some_text[BUFSIZ];
};

int main()
```

```
{
    int running = 1;
    int msgid, msgid2;
    struct my_msg_st some_data, send_data;
    long int msg_to_receive = 0;
    char buffer[BUFSIZ];

    send_data.my_msg_type = 1;

    msgid = msgget((key_t)1234, 0666 | IPC_CREAT);
    msgid2 = msgget((key_t)2345, 0666 | IPC_CREAT);

if (msgid == -1 || msgid2 == -1) {
    fprintf(stderr, "msgget failed with error: %d\n", errno);
    exit(EXIT_FAILURE);
}

while(running) {
    printf("\nWaiting for client message...\n");
    if (msgrcv(msgid, (void *)&some_data, BUFSIZ, msg_to_receive, 0) == -1){
        fprintf(stderr, "msgrcv failed with error: %d\n", errno);
        exit(EXIT_FAILURE);
}
printf("\nMessage received from client : %s", some_data.some_text);
if (strncmp(some_data.some_text, "stop", 4) == 0) {
    running = 0;
} else {
    printf("Enter a message from server:");
    fgets(buffer, BUFSIZ, stdin);
    send_data.my_msg_type = 1;
```

```
    strcpy(send_data.some_text, buffer);

    if (msgsnd(msgid2, (void *)&send_data, MAX_TEXT, 0) == -1) {
        fprintf(stderr, "msgsnd failed\n");
        exit(EXIT_FAILURE);
}
if (strncmp(buffer, "stop", 4) == 0) {
    running = 0;
}
}
}
if (msgctl(msgid, IPC_RMID, 0) == -1) {
    fprintf(stderr, "msgctl(IPC_RMID) failed\n");
    exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```

Output: -