TECHNO INDIA COLLEGE OF TECHNOLOGY

Department of Computer Science & Engineering

Computer Network Lab Manual

Paper Code: CS692

Objective: Implementation of Inter Process Communication using Message Queue.		
LAB 1	1. Understanding Inter Process Communication. 2. Implementation of Inter Process Communication using MessageQueue.	3 Hours

IPC Message Queue

In computer science, inter-process communication or interprocess communication (IPC) refers specifically to the mechanisms an operating system provides to allow the processes to manage shared data. Typically, applications can use IPC, categorized as clients and servers, where the client requests data and the server responds to client requests.

Message queue: A data stream which usually preserves message boundaries. Typically implemented by the operating system, they allow multiple processes to read and write to the message queue without being directly connected to each other. Message queues provide an asynchronous communications protocol, meaning that the sender and receiver of the message do not need to interact with the message queue at the same time. Messages placed onto the queue are stored until the recipient retrieves them.

[A "message boundary" is the separation between two messages being sent over a protocol. If you send "FOO" and then "BAR", the other end will receive two separate messages, one containing "FOO" and the other containing "BAR" and not "FOOBAR"]

IPC Message Queue Implementation in C

A simple implementation of IPC Message Queues.
IPC_msgq_send.c adds the message on the message queue .
IPC_msgq_rcv.c removes the message from the message queue.

To use this program first compile and run IPC_msgq_send.c to add a message to the message queue. To see the Message Queue type ipcs -q on your Unix/Linux Terminal.

```
vgupta> ipcs -q                                          This depicts that there is one
                                                         message in the queue
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x000004d2 9240578    vgupta80   666        12           1
```

Now compile and run IPC_msgq_rcv.c to read the message from the Message Queue.
To see that you have read the message again use ipcs -q

```
                                    The message has been removed
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x000004d2 9240578    vgupta80   666        0            0
```

```
//IPC_msgq_send.c

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAXSIZE     128

void die(char *s)
{
  perror(s);
  exit(1);
}

typedef struct msgbuf
{
    long    mtype; /* message type, must be > 0 */
    char    mtext[MAXSIZE];
};

main()
{
    int msqid;
    int msgflg = IPC_CREAT | 0666;
    key_t key;
    struct msgbuf sbuf;
    size_t buflen;

    key = 1234;

    if ((msqid = msgget(key, msgflg )) < 0)   //Get the message queue ID for
the given key
      die("msgget");

    //Message Type
    sbuf.mtype = 1;

    printf("Enter a message to add to message queue : ");
    scanf("%[^\n]",sbuf.mtext);
    getchar();

    buflen = strlen(sbuf.mtext) + 1 ;

    if (msgsnd(msqid, &sbuf, buflen, IPC_NOWAIT) < 0)
    {
        printf ("%d, %d, %s, %d\n", msqid, sbuf.mtype, sbuf.mtext, buflen);
        die("msgsnd");
    }

    else
        printf("Message Sent\n");

    exit(0);
}
```

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.

msgget(): either returns the message queue identifier for a newly created message queue or returns the identifiers for a queue which exists with the same key value.

msgsnd(): Data is placed on to a message queue by calling msgsnd().

msgrcv(): messages are retrieved from a queue.

ftok(): is use to generate a unique key.

msgctl(): It performs various operations on a queue. Generally it is use to destroy message queue.

IPC_CREAT | 0666 for a server (i.e., creating and granting read and write access to the server)

IPC_NOWAIT: If the message queue is full, then the message is not written to the queue, and control is returned to the calling process. If not specified, then the calling process will suspend (block) until the message can be written.

```
//IPC_msgq_rcv.c

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE     128

void die(char *s)
{
  perror(s);
  exit(1);
}

typedef struct msgbuf
{
    long    mtype;
    char    mtext[MAXSIZE];
} ;


main()
{
    int msqid;
    key_t key;
    struct msgbuf rcvbuffer;
```

```
    key = 1234;

    if ((msqid = msgget(key, 0666)) < 0)
      die("msgget()");


     //Receive an answer of message type 1.
    if (msgrcv(msqid, &rcvbuffer, MAXSIZE, 1, 0) < 0)
      die("msgrcv");

    printf("%s\n", rcvbuffer.mtext);
    exit(0);
}
```