Operating Systems Lab

Lab # 15

IPC using Message Queuing

and Shared Memory

Instructor: Engr. Muhammad Usman

Email: usman.rafiq@nu.edu.pk

Course Code: CL2006

Department of Computer Science,

National University of Computer and Emerging Sciences FAST

Peshawar Campus

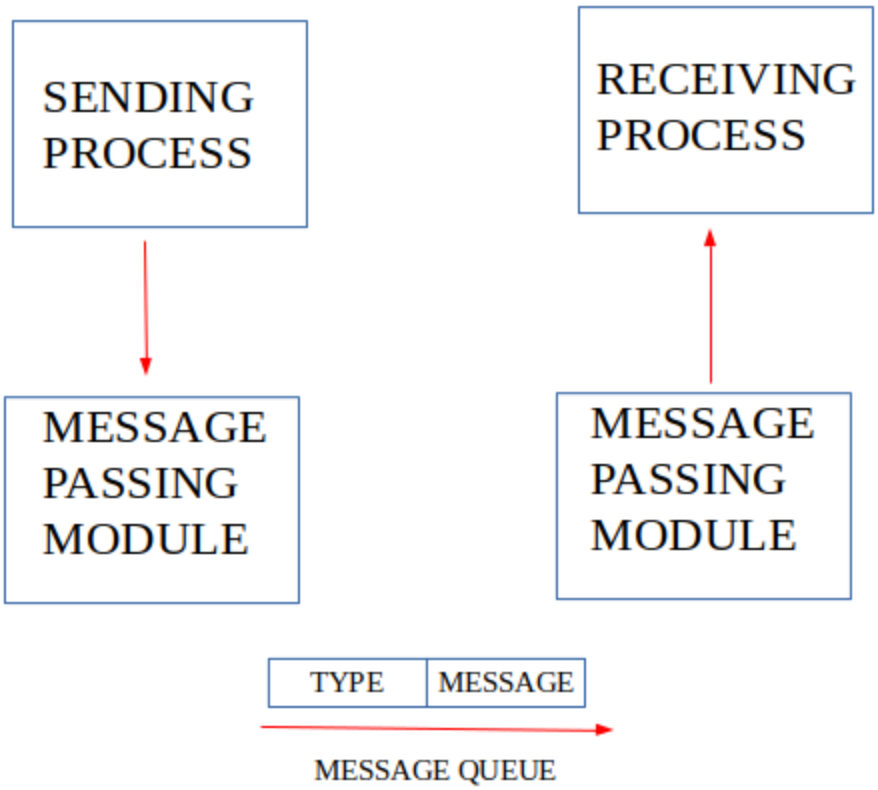
Table of Contents

IPC using Message Queues

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.



System calls used for message queues:

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

● 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.

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

MESSAGE QUEUE FOR WRITER PROCESS

```c
// C Program for Message Queue (Writer Process)
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MAX 10
// structure for message queue
struct mesg_buffer {
long mesg_type;
char mesg_text[100];
} message;
int main()
{
 key_t key;
 int msgid;
 // ftok to generate unique key
 key = ftok("progfile", 65);
 // msgget creates a message queue
 // and returns identifier
 msgid = msgget(key, 0666 | IPC_CREAT);
 message.mesg_type = 1;
 printf("Write Data : ");
 fgets(message.mesg_text,MAX,stdin);
 // msgsnd to send message
 msgsnd(msgid, &message, sizeof(message), 0);
 // display the message
 printf("Data send is : %s \n", message.mesg_text);
 return 0;
}
```

MESSAGE QUEUE FOR READER PROCESS

```c
// C Program for Message Queue (Reader Process)
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
// structure for message queue
struct mesg_buffer {
long mesg_type;
char mesg_text[100];
} message;

int main()
{
 key_t key;
 int msgid;
 // ftok to generate unique key
 key = ftok("progfile", 65);
 // msgget creates a message queue
 // and returns identifier
 msgid = msgget(key, 0666 | IPC_CREAT);
 // msgrcv to receive message
 msgrcv(msgid, &message, sizeof(message), 1, 0);
 // display the message
 printf("Data Received is : %s \n",
 message.mesg_text);
 // to destroy the message queue
 msgctl(msgid, IPC_RMID, NULL);
 return 0;
}
```

IPC through shared memory

Inter Process Communication through shared memory is a concept where two or more process can access the common memory. And communication is done via this shared memory where changes made by one process can be viewed by another process.

The problem with pipes, fifo and message queue – is that for two process to exchange information. The information has to go through the kernel.

● Server reads from the input file.

● The server writes this data in a message using either a pipe, fifo or message queue.

● The client reads the data from the IPC channel,again requiring the data to be copied from kernel's IPC buffer to the client's buffer.

● Finally the data is copied from the client's buffer.

A total of four copies of data are required (2 read and 2 write). So, shared memory provides a way by letting two or more processes share a memory segment. With Shared Memory the data is only copied twice – from input file into shared memory and from shared memory to the output file.

SYSTEM CALLS USED ARE:

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

● shmget(): int shmget(key_t,size_tsize,intshmflg); upon successful completion, shmget() returns an identifier for the shared memory segment.

● shmat(): Before you can use a shared memory segment, you have to attach yourself

to it using shmat(). void *shmat(int shmid ,void *shmaddr ,int shmflg);

● shmid is shared memory id. shmaddr specifies specific address to use but we

should set

it to zero and OS will automatically choose the address.

● shmdt(): When you're done with the shared memory segment, your program

should detach itself from it using shmdt(). int shmdt(void *shmaddr);

● shmctl(): when you detach from shared memory,it is not destroyed. So, to destroy

● shmctl() is used. shmctl(int shmid,IPC_RMID,NULL);

SHARED MEMORY FOR WRITER PROCESS:

```c
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>

int main()
{
    // ftok to generate unique key
    key_t key = ftok("shmfile",65);
    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);
    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,(void*)0,0);
    printf("Write Data : ");
    gets(str);
    printf("Data written in memory: %s\n",str);
    //detach from shared memory
    shmdt(str);
    return 0;
}
```

SHARED MEMORY FOR READER PROCESS:

```c
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
int main()
{
    // ftok to generate unique key
    key_t key = ftok("shmfile",65);
    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);
    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,(void*)0,0);
    printf("Data read from memory: %s\n",str);
    //detach from shared memory
    shmdt(str);
    // destroy the shared memory
    shmctl(shmid,IPC_RMID,NULL);
    return 0;
}
```

Reference

https://www.geeksforgeeks.org/ipc-shared-memory/

https://www.geeksforgeeks.org/ipc-using-message-queues/?ref=lbp