College of Engineering,Trivandrum

Experiment 7

Network Programming Lab

Author:
Alan Anto

Registration Number :
TVE16CS09

February 20, 2019

Contents

1 Pipes,Message Queue and Shared Memory

1.1 AIM

Implement programs for Inter Process Communication using PIPE, Message Queue and Shared Memory.

1.2 Theory

1.2.1 Pipes

A pipe acts as a conduit allowing two processes to communicate. Pipes were one of the first IPC mechanisms in early UNIX systems. They typically provide one of the simpler ways for processes to communicate with one another, although they also have some limitations. Two common types of pipes used on both UNIX and Windows systems: ordinary pipes and named pipes. Named pipes can be used to provide communication between processes on the same computer or between processes on different computers across a network, as in case of a distributed system. The Ordinary pipe in Operating Systems allows the two procedures to communicate in a standard way: the procedure writes on the one end (the write end) and reads on the consumer side or another end (the read-end).

1.2.2 Oridinary Pipes

Ordinary pipes allow two processes to communicate in standard producer consumer fashion: the producer writes to one end of the pipe (the write-end) and the consumer reads from the other end (the read-end). Ordinary pipes are unidirectional. If two-way communication is required, two pipes must be used, with each pipe sending data in a different direction. On UNIX systems, ordinary pipes are constructed using the function

```
pipe( int fd[ ] )
```

1.2.3 Named Pipes

Named pipes provide a much more powerful communication tool. Com- munication can be bidirectional, and no parent-child relationship is required. Once a named pipe is established, several processes can use it for communication. In fact, in a typical

scenario, a named pipe has several writers. Additionally, named pipes continue to exist after communicating processes have finished.

1.2.4 Message Queues

A message queue can be created by one process and used by multiple processes that read and/or write messages to the queue.Message queues have internal structure. With a named pipe, a writer is just pumping bits. For a reader, there's no distinction between different calls to write() from different writers.Message queues are priority-driven.

1.2.5 Shared Memory

Interprocess communication using shared memory requires communicating processes to establish a region of shared memory. Typically, a shared-memory region resides in the address space of the process creating the shared-memory segment. Other processes that wish to communicate using this shared-memory segment must attach it to their address space. Recall that, normally, the operating system tries to prevent one process from accessing another processs memory. Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas. The form of the data and the location are determined by these processes and are not under the operating systems control. The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

1.3 Program

1.3.1 Pipes

```
#include<stdio.h>
#include<unistd.h>

int main() {
    int pipefds[2];
    int returnstatus;
    char writemessages[2][20]={"Hi", "Hello"};
    char readmessage[20];
    returnstatus = pipe(pipefds);
```

```
    if (returnstatus == -1) {
       printf("Unable to create pipe\n");
       return 1;
    }

    printf("Writing to pipe - Message 1 is %s\n", writemessages[0]);
    write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
    read(pipefds[0], readmessage, sizeof(readmessage));
    printf("Reading from pipe Message 1 is %s\n", readmessage);
    printf("Writing to pipe - Message 2 is %s\n", writemessages[0]);
    write(pipefds[1], writemessages[1], sizeof(writemessages[0]));
    read(pipefds[0], readmessage, sizeof(readmessage));
    printf("Reading from pipe  Message 2 is %s\n", readmessage);
    return 0;
}
```

1.3.2 Message Queue

```
#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;
    key = ftok("progfile", 65);


    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_type = 1;

    printf("Write Data : ");
    gets(message.mesg_text);
```

```
    msgsnd(msgid, &message, sizeof(message), 0);
    printf("Data send is : %s \n", message.mesg_text);

    return 0;
}
```

1.3.3 Shared memory

```
#include<iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;

int main()
{
    key_t key = ftok("shmfile",65);

    int shmid = shmget(key,1024,0666|IPC_CREAT);

    char *str = (char*) shmat(shmid,(void*)0,0);

    cout<<"Write Data : ";
    gets(str);

    printf("Data written in memory: %s\n",str);

    shmdt(str);

    return 0;
}
```

1.4 Output

1.4.1 Pipes

```
Alans-MacBook-Air:NPLab alan$ ./a.out
Writing to pipe - Message 1 is Hi
Reading from pipe - Message 1 is Hi
Writing to pipe - Message 2 is Hi
Reading from pipe - Message 2 is Hello
Alans-MacBook-Air:NPLab alan$
```

1.4.2 Message Queue

```
Alans-MacBook-Air:NPLab alan$ gcc msg.c
Alans-MacBook-Air:NPLab alan$ ./a.out
warning: this program uses gets(), which is unsafe.
Write Data : hi
Data send is : hi
Alans-MacBook-Air:NPLab alan$
```

1.4.3 Shared Memory

```
Alans-MacBook-Air:NPLab alan$ g++ sharedmem.cpp
Alans-MacBook-Air:NPLab alan$ ./a.out
warning: this program uses gets(), which is unsafe.
Write Data : hello
Data written in memory: hello
Alans-MacBook-Air:NPLab alan$
```

1.5 Result

Interprocess communication was implemented using pipes,message queue and Shared memory and the output was verified.