Networking Lab Assignment 7

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

Albin Antony

19 February 2019

1 Inter Process Communication

1.1 Aim

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

1.2 Theory

1.2.1 Pipe

pipe() is used for passing information from one process to another. pipe() is unidirectional therefore, for two-way communication between processes, two pipes can be set up, one for each direction.

1.2.2 Named Pipe

Named Pipes : Named pipes provide a much more powerful communication tool. Communication 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.

1.2.3 Message Queues

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. Message queues have implicit or explicit limits on the size of data that may be transmitted in a single message and the number of messages that may remain outstanding on the queue.

1.2.4 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 anther process. 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.

1.3 Program

1.3.1 Pipe

```
#include<iostream>
#include <sys/types.h>
#include<unistd.h>
using namespace std;

int main(){
    int pid,pip[2];
    char string[20];
    pipe(pip);

    pid =fork();
    if(pid==0){
        write(pip[1],"hello\0",6);
    }
    else{
        read(pip[0],string,6);
        printf("%s",string);
    }
}
```

Named Pipe

Writer

```
#include<iostream>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<string.h>
#include<unistd.h>

using namespace std;
int main(){
    int fd;
    mkfifo("fifo",0666);
    char arr[10],arr1[10];
    while(1){
        fd=open("fifo",O_WRONLY);
        fgets(arr, 10, stdin);
        write(fd,arr,strlen(arr));
        close(fd);

    }
```

```
}
```

Reader

```
#include<iostream>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<string.h>
#include<unistd.h>

using namespace std;
int main(){
    int fd;
    mkfifo("fifo",0666);
    char arr[10],arr1[10];
    while(1){
        fd=open("fifo",O_RDONLY);
        read(fd,arr,10);
        printf("%s",arr);
        close(fd);
    }
}
```

1.3.2 Message Queue

Writer

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

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;
}
```

Reader

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

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);
        msgrcv(msgid, &message, sizeof(message), 1, 0);
        printf("Data Received is : %s \n",message.mesg_text)
            ;
        msgctl(msgid, IPC_RMID, NULL);

        return 0;
}
```

1.3.3 Shared Memory

Writer

```
#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);

        printf("Data read from memory: %s\n",str);

        shmdt(str);
        shmctl(shmid,IPC_RMID,NULL);

        return 0;
}
```

Reader

```
#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 : ";
        cin>>str;

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

        return 0;
}
```

1.4 Output

1.4.1 Pipe

```
$ g++ pipe.cpp -lpthread
$ ./a.out
hello from child
```

1.4.2 Named Pipe

```
$ g++ npipeReader.cpp -o reader
```

```
$ ./reader
hello
hey

$ g++ npipeWriter.cpp -o writer
$ ./writer
hello
hey
```

1.4.3 Message Queue

```
$ gcc msgWriter.c -o w
$ ./w
Write Data : hello
Data send is : hello

$ gcc msgReader.c -o r
$ ./r
Data Received is : hello
```

1.4.4 Shared Memory

```
$ g++ sharedMemWriter.cpp -o w
$ ./w
Write Data : hello
Data written in memory: hello

$ g++ sharedMemReader.cpp -o r
$ ./r
Data read from memory: hello
```

1.5 Result

Implemented programs for IPC using pipes,Message queue and Shared Memory in C++ compiled on g++ 6.3.0 and executed on Debian 4.9 Kernel 4.9 and outputs were verified.