INTERPROCESS COMMUNICATION USING PIPES, MESSAGE QUEUE AND SHARED MEMORY

Network Programming Lab (CS334)

April 22, 2021

Sheen Xavier A
Roll No : 57
University Roll No : TVE18CS058
Department of Computer Science and Engineering
College of Engineering, Trivandrum

Contents

Interprocess Communication using Pipes, Message Queue and Shared Memory

1.1 Aim

To implement programs for Inter Process Communication using PIPE, Message Queue and Shared Memory

1.2 Theory

Pipes

Pipe is a communication medium between two or more related or interrelated processes. It can be either within one process or a communication between the child and the parent processes, or even be multi-level such as communication between the parent, child, grand-child, etc. Two common types of pipes used on both UNIX and Windows systems : ordinary pipes and named pipes.

Ordinary Pipes

Ordinary pipes allow two processes to communicate in a producer consumer fashion where the producer writes to one end of the pipe and the consumer reads from the other end. They are unidirectional. On UNIX systems, ordinary pipes are constructed using the function pipe(fd), where fd[0] is the read-end of the pipe and fd[1] is the write-end. Here

fd is two-element array. The major limitation of an ordinary pipe is that they are always local, ie. they cannot be used for communication over a network.

Named Pipes

A named pipe is a more powerful communication tool as it can be bidirectional and does not require any parent-child relationship. Once a named pipe is established, several processes can use it for communication. A named pipe exists in the file system. After input-output has been performed by the sharing processes, the pipe still exists in the file system independently of the process, and can be used for communication between some other processes. In order to create a named pipe we make use of the mkfifo() function. A FIFO special file is entered into the file system by calling it. Once we have created it, any process can open it for reading or writing from both ends.

Message Queues

It is a component used for IPC, or for inter-thread communication within the same process. It makes use of a queue for messaging – the passing of control or of content. Message queue is a linked list of messages stored within the kernel and identified by a message queue identifier. 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. A new queue is created or an existing queue opened by msgget(). New messages are added to the end of a queue by msgsnd() and are fetched from the queue by msgrcv(). We don't have to fetch the messages in a FIFO manner. Instead, we can fetch messages based on their type field.

Shared Memory

Shared memory is a component used in IPC, where two or more process can access the common memory. Here, communication is done via this shared memory where changes made by one process can be viewed by anther process. The function shmget() is used to create and return an identifier for the shared memory segment. Before one can use a shared memory segment, we should attach to it using the function, shmat(). Ususally the memory address to be used is automatically chosen by the OS. After this, we can use the memory segment for communication. When we are done with the shared memory segment, the program should detach itself from it using shmdt(). In order to destroy the shared memory segment, shmctl() is used.

1.3 Programs

1.3.1 IPC using Ordinary Pipes

Description

Program to demonstrate interprocess communication between a child process and its parent process with the use of Ordinary Pipes.

Source Code

```
// Name   : cpcomm.c
// Desc   : Program which simulates the communication between
//          parent and child process via ordinary pipes
// Input  : Data to be transmitted
// Output : Communication between the parent and child process
// Author : Sheen Xavier A
// Date   : 22/04/2021

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

int main()
{
    int dp[2], id;
    char msgcp[30];
    // Data to be transmitted
    char *data = "Successor sends his regards!\n";
    // Creating the pipe
    if(pipe(dp) == -1)
    {
        // Exiting incase of failure in creating pipe
        printf("Pipe creation failure!\n");
        exit(0);
    }
    // Creating the child process
    id = fork();

    // Child Process Handling Section
    if(id == 0)
```

```
31      {
32          // A short sleep to show delay
33          sleep(1);
34          printf("Child writing the data to be sent to parent...\n");
35          // Child process writes data into the pipe
36          write(dp[1], data, 30);
37      }
38
39      // Parent Process Handling Section
40      else
41      {
42          // A short sleep to show delay
43          sleep(1);
44          // Parent process reads data from the pipe
45          read(dp[0], msgcp, 30);
46          printf("Parent reading data from the child...\n");
47          printf("Message from Child to Parent : %s\n", msgcp);
48      }
49
50      return 0;
51  }
```

Output

```
sheenxavi004@Beta-Station:~$ cc cpcomm.c
sheenxavi004@Beta-Station:~$ ./a.out
Child writing the data to be sent to parent...
Parent reading data from the child...
Message from Child to Parent : Successor sends his regards!
```

Figure 1.1: Communication between parent and child processes using ordinary pipe

1.3.2 IPC using Named Pipes

Description

Program to demonstrate interprocess communication between processes using Named Pipes.

Source Code

```
// Name   : npipewrite.c
// Desc   : Program to demonstrate IPC using named pipes
//          This program writes first and then reads
// Input  : Data to be transmitted
// Output : Communication between two process
// Author : Sheen Xavier A
// Date   : 22/04/2021

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

int main()
{
    int fd;

    // FIFO File Path
    char * spfifo = "/tmp/ipc";
    char rdata[100], wdata[100];
    // Creating the FIFO file for IPC using mkfifo()
    // Setting permission for the FIFO File as read-write
    mkfifo(spfifo, 0666);

    printf("Dave Terminal\n");
    while(1)
    {
        // Opening the FIFO for write only
        fd = open(spfifo, O_WRONLY);

        printf("Type Message : ");
```

```
34          scanf(" %[^\n]", wdata); // Reading input from the user
35          write(fd, wdata, strlen(wdata) + 1); //Writing the input onto
    FIFO
36
37          // Opening the FIFO for read only
38          fd = open(spfifo, O_RDONLY);
39
40          read(fd, rdata, 100); // Reading content from FIFO
41          printf("HAL-9000 says : %s\n", rdata); // Displaying the
    message from other process
42
43          close(fd); //Closing the FIFO
44      }
45      return 0;
46  }
47
48  //*********************************************************************
49
50  // Name    : npiperead.c
51  // Desc    : Program to demonstrate IPC using named pipes
52  //           This program reads first and then writes
53  // Input   : Data to be transmitted
54  // Output  : Communication between two process
55  // Author  : Sheen Xavier A
56  // Date    : 22/04/2021
57
58  #include <stdio.h>
59  #include <unistd.h>
60  #include <string.h>
61  #include <fcntl.h>
62  #include <sys/stat.h>
63  #include <sys/types.h>
64
65  int main()
66  {
67      int fd;
68
69      // FIFO File Path
70      char * spfifo = "/tmp/ipc";
71      char rdata[100], wdata[100];
72      // Creating the FIFO file for IPC using mkfifo()
73      // Setting permission for the FIFO File as read-write
```

```
74      mkfifo(spfifo, 0666);
75
76      printf("HAL-9000 Terminal\n");
77      while(1)
78      {
79          // Opening the FIFO for read only
80          fd = open(spfifo, O_RDONLY);
81
82          read(fd, rdata, 100); // Reading content from FIFO
83          printf("Dave says : %s\n", rdata); // Displaying the message
    from other process
84
85          // Opening the FIFO for write only
86          fd = open(spfifo, O_WRONLY);
87
88          printf("Type Message : ");
89          scanf(" %[^\n]", wdata); // Reading input from the user
90          write(fd, wdata, strlen(wdata) + 1); //Writing the input onto
    FIFO
91
92          close(fd); //Closing the FIFO
93      }
94      return 0;
95  }
```

Output

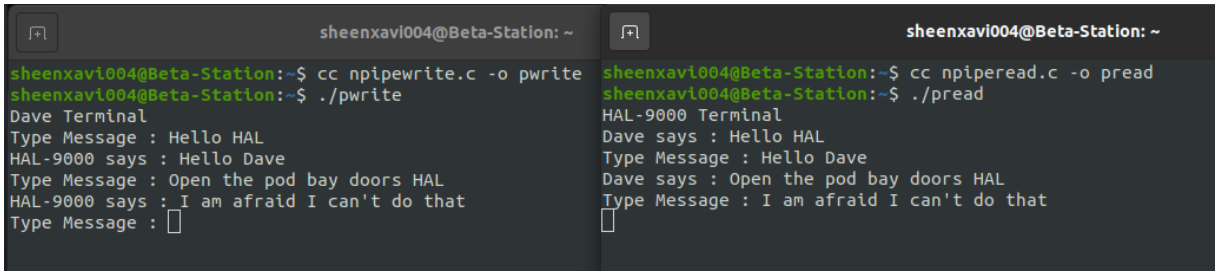

Figure 1.2: IPC between Dave and HAL-9000 processes using named pipe

1.3.3 IPC using Memory Queue

Description

Program to demonstrate interprocess communication between reader and writer processes with the help of message queue.

Source Code

```c
// Name    : msgwrite.c
// Desc    : Program to demonstrate IPC using Message Queue
//           This program is used to write data into the Message Queue
// Input   : Data to be transmitted
// Output  : Data that was transmitted
// Author  : Sheen Xavier A
// Date    : 22/04/2021

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

// Data structure for message queue
struct msg_buff {
    long msg_type;
    char msg_data[100];
}msg;

int main()
{
    key_t key; // Varible for storing unique key
    int msgid;

    // Generating the unique key
    key = ftok("msgfile", 65);

    // Creates a message queue and stores
    // the generated id in msgid
    msgid = msgget(key, 0666 | IPC_CREAT);
    msg.msg_type = 1; // Setting message type as 1

    printf("Message to be transmitted : ");
    scanf(" %[^\n]", msg.msg_data); // Reading the message to be send
```

```c

    // Transmitted message
    printf("Successfully relayed the message : %s\n", msg.msg_data);

    msgsnd(msgid, &msg, sizeof(msg), 0); // Sending the message
    contents

    return 0;
}

//*********************************************************************

// Name    : msgread.c
// Desc    : Program to demonstrate IPC using Message Queue
//           This program is used to read data from the Message Queue
// Input   : None
// Output  : Data that was transmitted
// Author  : Sheen Xavier A
// Date    : 22/04/2021

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

// Data structure for message queue
struct msg_buff {
    long msg_type;
    char msg_data[100];
}msg;

int main()
{
    key_t key; // Varible for storing unique key
    int msgid;

    // Generating the unique key
    key = ftok("msfile", 65);

    // Creates a message queue and stores
    // the generated id in msgid
    msgid = msgget(key, 0666 | IPC_CREAT);
    msg.msg_type = 1; // Setting message type as 1
```

```
75
76      msgrcv(msgid, &msg, sizeof(msg), 1, 0); // Recieving the
        transmitted message
77
78      printf("Data Recieved : %s\n", msg.msg_data);
79
80      //Destroying the message queue
81      msgctl(msgid, IPC_RMID, NULL);
82
83      return 0;
84  }
```

Output

```
sheenxavi004@Beta-Station:~$ cc msgwrite.c
sheenxavi004@Beta-Station:~$ ./a.out
Message to be transmitted : FBI! Freeze!
Successfully relayed the message : FBI! Freeze!
sheenxavi004@Beta-Station:~$ cc msgread.c
sheenxavi004@Beta-Station:~$ ./a.out
Data Recieved : FBI! Freeze!
```

Figure 1.3: IPC between writer and reader processes using message queue

1.3.4 IPC using Shared Memory

Description

Program to demonstrate interprocess communication between reader and writer processes with the help of shared memory.

Source Code

```
1  // Name    : shwrite.c
2  // Desc    : Program to demonstrate IPC using Shared Memory
3  //           This program is used to write data into the Shared Memory
4  // Input   : Data to be transmitted
5  // Output  : Data that was transmitted
6  // Author  : Sheen Xavier A
7  // Date    : 22/04/2021
8
9  #include <stdio.h>
```

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

int main()
{
    // Generates the unique which is stored into key
    key_t key = ftok("shared", 65);

    // Creates a shared memory and stores the generated
    // id in memid
    int memid = shmget(key, 1024, 0666 | IPC_CREAT);

    // Attaching string data to the shared memory segment
    char *data = (char *)shmat(memid, (void*)0, 0);

    printf("Data to be transmitted : ");
    scanf(" %[^\n]", data); // Writing data to be transmitted

    // Transmitted message
    printf("Successfully relayed the message : %s\n", data);

    // After writing, we detach data from shared memory
    shmdt(data);

    return 0;
}

//*********************************************************************

// Name    : shread.c
// Desc    : Program to demonstrate IPC using Shared Memory
//           This program is used to read data from the Shared Memory
// Input   : None
// Output  : Data that was transmitted
// Author  : Sheen Xavier A
// Date    : 22/04/2021

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

int main()
```

```c
{
    // Generates the unique which is stored into key
    key_t key = ftok("shared", 65);

    // Creates a shared memory and stores the generated
    // id in memid
    int memid = shmget(key, 1024, 0666 | IPC_CREAT);

    // Attaching string data to the shared memory segment
    char *data = (char *)shmat(memid, (void*)0, 0);

    printf("Data Recieved : %s\n", data); // Reading transmitted data

    // After reading, we detach data from shared memory
    shmdt(data);

    // Destroying the shared memory
    shmctl(memid, IPC_RMID, NULL);

    return 0;
}
```

Output

```
sheenxavi004@Beta-Station:~$ cc shwrite.c
sheenxavi004@Beta-Station:~$ ./a.out
Data to be transmitted : Here's Johnny!
Successfully relayed the message : Here's Johnny!
sheenxavi004@Beta-Station:~$ cc shread.c
sheenxavi004@Beta-Station:~$ ./a.out
Data Recieved : Here's Johnny!
```

Figure 1.4: IPC between writer and reader processes using shared memory

1.4 Result

The programs were developed as per requirement and the outputs were verified. It was also tested against various test cases.