6 PIPE,Message Queue and Shared Memory

6.1 Aim

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

6.2 Theory

6.2.1 PIPE

- **Ordinary PIPE**
  PIPE is a method for interprocess communication.One end of the pipe is used for writing data and the other end of the pipe is used for reading data.The communication is unidirectional and the pipe is terminated when the process exits.

- **Named PIPE**
  In named PIPE, a file is used for storing data from one process which is to be read by another process.Since a file is used,a named PIPE can exist beyond the lifetime of a process.The communication in a named PIPE is bidirectional.

6.2.2 Message Queue

Message queue is a method for interprocess communication in which a linked list stored in the kernel is used by processes to transfer data between processes.Each message queue will have a unique key which is used to identify the queue.So only processes which have these unique key will be able to use the message queue.Bidirectional communication is possible in message queue.

6.2.3 Shared Memory

In interprocess communication using shared memory, a memory space is shared between processes for communication.The shared memory space resides in the address space of the process that creates the shared memory space.Processes which wish to use this memory space must attach the shared memory to its address space.One process can write to the memory location and the other process can read from the memory location.

6.3 Algorithm

Algorithm 1 Ordinary PIPE

```
procedure MAIN PROCEDURE
    if pipe(p) < 0 then
        Throw error and exit
    end if
    if (pid = fork()) > 0 then                          ▷ The process is parent
        Write the messages to the pipe and wait for child process to terminate
    else                                                ▷ The process is child
        read the messages from the pipe,print the messages and exit
    end if
end procedure
```

Algorithm 2 Named Pipe Writer

```
procedure MAIN
    mkfifo(path,0666)                    ▷ Creates the pipe if it does not exist
    while true do
        fd1 = open(path,WRONLY)          ▷ Open the pipe in write only mode
        fgets(str2, 80, stdin)
        write(fd1, str2, strlen(str2)+1)          ▷ write the data into the pipe
        close(fd1)                                ▷ close the file descriptor
        fd1 = open(path,RDONLY)          ▷ Open the pipe in read only mode
        read(fd1, str1, 80)   ▷ read the data from the pipe and print the data
        printf("Reader: % s \n", str1)
        close(fd1)                                ▷ close the file descriptor
    end while
end procedure
```

Algorithm 3 Named Pipe Reader

```
procedure MAIN
    mkfifo(path,0666)                          ▷ Creates the pipe if it does not exist
    while true do
        fd1 = open(path,RDONLY)                ▷ Open the pipe in read only mode
        read(fd1, str1, 80)   ▷ read the data from the pipe and print the data
        printf("Writer: %s \n", str1)
        close(fd1)                                          ▷ close the file descriptor
        fd1 = open(path,WRONLY)                ▷ Open the pipe in write only mode
        fgets(str2, 80, stdin)
        write(fd1, str2, strlen(str2)+1)                ▷ write the data into the pipe
        close(fd1)                                          ▷ close the file descriptor
    end while
end procedure
```

Algorithm 4 Message Queue Writer

```
procedure MAIN
    key =ftok("progfile",65)    ▷ Creates a unique key for the message queue
    msgid = msgget(key,0666|IPC_CREAT)  ▷ Creates the message queue if
it doesnot exist with key
    message.mesg_type = 1
    printf("Enter the data to be written ")
    scanf("%s",message.mesg_text)
    msgsnd(msgid,& message,sizeof(message),0) ▷ Sends data to the message
queue
    printf("Data send is % s \n",message.mesg_text)
end procedure
```

Algorithm 5 Message Queue Reader

```
procedure MAIN
    key = ftok("progfile",65)   ▷ Creates a unique key for the message queue
    msgid = msgget(key,0666 | IPC_CREAT)▷ Creates the message queue if
it doesnot exist
    msgrcv(msgid,&message,sizeof(message),1,0)    ▷ Read message from the
queue
    printf("The data recieved is %s \n",message.mesg_text)
    msgctl(msgid,IPC_RMID,NULL)                        ▷ Destroys the queue
end procedure
```

Algorithm 6 Shared Memory Writer

procedure MAIN
key_t key = ftok("shmfile",65) ▷ Create a unique key for the shared memory
int shmid = shmget(key,1024,0666 | IPC_CREAT) ▷ Create the shared memory location if it does not exist
char * str = (char *)shmat(shmid,(void *)0,0) ▷ Attach a variable to the shared memory
printf("Enter the data to be written ")
scanf("%s",str) ▷ Write data to the memory location
printf("The data written to the shared memory is %s",str)
shmdt(str) ▷ Destroys the variable attached to the shared memory
end procedure

Algorithm 7 Shared Memory Reader

procedure MAIN
key_t key = ftok("shmfile",65) ▷ Create a unique key for the shared memory
int shmid = shmget(key,1024,0666 | IPC_CREAT) ▷ Create the shared memory location if it does not exist
char * str = (char *)shmat(shmid,(void *)0,0) ▷ Attach a variable to the shared memory
printf("The data read from shared memory is %s \n",str) ▷ Print data from shared memory
shmdt(str) ▷ Destroy the variable attached to the shared memory
shmctl(shmid,IPC_RMID,NULL) ▷ Destroy the shared memory
end procedure

6.4 Code

PIPE

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/wait.h>
#define MSGSIZE 16
char msg[][MSGSIZE] = {"hello, world #1","hello, world #2","hello, world #3"};

int main()
{
  char inbuf[MSGSIZE];
  int p[2], i,pid;

  if (pipe(p) < 0)
    exit(1);
  if((pid = fork()) > 0){
```

```
    for(i = 0;i<3;i++){
      printf("Writing message %s to pipe ...\n",msg[i],i);
      write(p[1],msg[i],MSGSIZE);
    }
    wait(NULL);
  }
  else{
    for (i = 0; i < 3; i++) {
      read(p[0], inbuf, MSGSIZE);
      printf("The parent says % s\n", inbuf);
    }
    printf("Finished reading");
    exit(2);
  }
  return 0;
}
```

Named Pipe Writer Process

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

int main()
{
    int fd;
    char * myfifo = "/tmp/myfifo";
    mkfifo(myfifo, 0666);
    char arr1[80], arr2[80];
    while (1)
    {
        fd = open(myfifo, O_WRONLY);
        fgets(arr2, 80, stdin);
        write(fd, arr2, strlen(arr2)+1);
        close(fd);

        fd = open(myfifo, O_RDONLY);
        read(fd, arr1, sizeof(arr1));
        printf("User2: %s\n", arr1);
        close(fd);
    }
    return 0;
}
```

Named Pipe Reader Process

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

```c
#include <sys/types.h>
#include <unistd.h>
int main()
{
    int fd1;
    char * myfifo = "/tmp/myfifo";
    mkfifo(myfifo, 0666);

    char str1[80], str2[80];
    while (1)
    {
        fd1 = open(myfifo,O_RDONLY);
        read(fd1, str1, 80);
        printf("User1: %s\n", str1);
        close(fd1);
        fd1 = open(myfifo,O_WRONLY);
        fgets(str2, 80, stdin);
        write(fd1, str2, strlen(str2)+1);
        close(fd1);
    }
    return 0;
}
```

Message Queue Writer Process

```c
#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("Enter the  data to be written ");
    scanf("%s",message.mesg_text);

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

    return 0;
}
```

Message Queue Reader Process

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

```c
#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("The data recieved is %s \n",message.mesg_text);
    msgctl(msgid,IPC_RMID,NULL);

    return 0;
}
```

Shared Memory Writer Process

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

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("Enter the data to be written ");
    scanf("%s",str);
    printf("The data written to the shared memory is %s",str);
    shmdt(str);

    return 0;
}
```

Shared Memory Reader Process

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

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("The data read from shared memory is %s \n",str);
    shmdt(str);
    shmctl(shmid,IPC_RMID,NULL);

```

```
13      return 0;
14  }
```

6.5 Output

- **Ordinary PIPE**

```
Writing message hello, world #1 to pipe ...
Writing message hello, world #2 to pipe ...
Writing message hello, world #3 to pipe ...
The parent says hello, world #1
The parent says hello, world #2
The parent says hello, world #3
```

- **Named PIPE Writer**

```
helloworld
User2: helloworld1
helloworld2
User2: helloworld3
```

- **Named PIPE Reader**

```
User1: helloworld
helloworld1
User1: helloworld2
helloworld3
```

- **Message Queue Writer**

```
Enter the  data to be written helloworld1
Data send is helloworld1
```

- **Message Queue Reader**

```
The data recieved is helloworld1
```

- **Shared Memory Writer**

```
Enter the data to be written helloworld123
The data written to the shared memory is helloworld123
```

- **Shared Memory Reader**

```
The data read from shared memory is helloworld123
```

6.6 Result

- **Ordinary PIPE**
  Ordinary PIPE was implemented in C++.The program creates a child reader process using fork while the parent acts as th writer process.The parent writes data to the pipe which is read by the child process.

- **Named PIPE**
  Named PIPE was implemented in C++.There are two programs, a reader program and a writer program.The reader and writer program creates the named PIPE if it does not exist.The writer PIPE takes user input and writes it to the queue.It then reads the data written by reader program from PIPE and prints the data.The reader process first reads the data from Named PIPE and displays it.It then takes user input and then writes to the named PIPE.

- **Message Queue**
  Message Queue was implemented in C++.There are two programs,a reader program and a writer program.The reader and writer program creates a message queue if it does not exist.The writer program takes a user input and writes it to the message queue.The reader program reads the data from the message queue and prints it.The reader program then closes the message queue

- **Shared Memory**
  Shared Memory was implemented in C++.There are two programs,a reader program and writer program.The reader program and writer program creates a shared memory if it does not exist.The writer program attaches a variable to the shared memory location.It then takes a user input to this variable.This is saved in shared memory.The reader program attaches a variable to the shared memory and prints the variable.