# 5.Write C programs to illustrate the following IPC mechanisms

* 1. **Pipes b) FIFOs c) Message Queues d) Shared Memory**

# Pipes DESCRIPTION:

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. Communication can also be multi-level such as communication between the parent, the child and the grand-child, etc. Communication is achieved by one process writing into the pipe and other reading from the pipe. To achieve the pipe system call, create two files, one to write into the file and another to read from the file.

# PROGRAM:

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

}

# OUTPUT:

student@KITS:~/oslab$ cc pipe.c student@KITS:~/oslab$ ./a.out Writing to pipe - Message 1 is Hi Reading from pipe – Message 1 is Hi Writing to pipe - Message 2 is Hello Reading from pipe – Message 2 is Hello student@KITS:~/oslab$

# FIFOs DESCRIPTION:

Pipes were meant for communication between related processes. Can we use pipes for unrelated process communication, say, we want to execute client program from one terminal and the server program from another terminal? The answer is No. Then how can we achieve unrelated processes communication, the simple answer is Named Pipes. Even though this works for related processes, it gives no meaning to use the named pipes for related process communication.

# PROGRAM:

**fifoclient.c**

#include<stdio.h> #include<fcntl.h> #include<stdlib.h> #include<unistd.h> int main()

{

FILE \*file1;

int fifo\_server,fifo\_client; char str[256];

char \*buf; int choice=1;

printf("Choose the request to be sent to server from options below"); printf("\n\t\t Enter 1 for O.S.Name \n \

Enter 2 for Distribution \n \ Enter 3 for Kernel version\n");

scanf("%d",&choice); fifo\_server=open("fifo\_server",O\_RDWR); if(fifo\_server < 0) {

printf("Error in opening file");

exit(-1);

}

write(fifo\_server,&choice,sizeof(int));

fifo\_client=open("fifo\_client",O\_RDWR);

if(fifo\_client < 0) { printf("Error in opening file"); exit(-1);

}

buf=malloc(10\*sizeof(char));

read (fifo\_client,buf,10\*sizeof(char));

printf("\n \*\*\*Reply from server is %s\*\*\*\n",buf); close(fifo\_server);

close(fifo\_client); return 0;

}

**fifoserver.c**

**PROGRAM:**

#include<stdio.h> #include<fcntl.h> #include<unistd.h> int main( )

{

FILE \*file1;

int fifo\_server,fifo\_client; int choice;

char \*buf;

fifo\_server = open("fifo\_server",O\_RDWR); if(fifo\_server<1) {

printf("Error opening file");

}

read(fifo\_server,&choice,sizeof(int)); sleep(10);

fifo\_client = open("fifo\_client",O\_RDWR); if(fifo\_server<1) {

printf("Error opening file");

}

switch(choice) { case 1:

buf="Linux"; write(fifo\_client,buf,10\*sizeof(char)); printf("\n Data sent to client \n"); break;

case 2:

buf="Fedora"; write(fifo\_client,buf,10\*sizeof(char));

printf("\nData sent to client\n"); break;

case 3:

buf="2.6.32";

write(fifo\_client,buf,10\*sizeof(char)); printf("\nData sent to client\n");

}

close(fifo\_server); close(fifo\_client);

}

# OUTPUT:

**TERMINAL-I**

student@KITS:~/oslab$ cc fifoclient.c student@KITS:~/oslab$ ./a.out

Choose the request to be sent to server from options below Enter 1 for O.S.Name

Enter 2 for Distribution Enter 3 for Kernel version 1

\*\*\*Reply from server is Linux\*\*\*

# TERMINAL-II

student@KITS:~/oslab$ cc fifoserver.c

student@KITS:~/oslab$ ./a.out Data sent to client

1. **Message Queues:**

# DESCRIPTION:

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.

# PROGRAM:

**Sender.c**

#include<stdio.h> #include<sys/ipc.h> #include<sys/msg.h> #include<sys/types.h> #include<stdlib.h> #define SIZE 2000 void main()

{

int mfd,mfd2,mfd3; struct

{

double mtype; char mtext[2000];

}s1,s2,s3; if((mfd=msgget(1000,IPC\_CREAT|0666))==-1)

{

perror("msgget:"); exit(1);

}

s1.mtype=1;

sprintf(s1.mtext,"%s","Hi friends... My name is message1"); if(msgsnd(mfd,&s1,1000,0)==-1)

{

perror("msgsnd"); exit(1);

}

if((mfd2=msgget(1000,IPC\_CREAT|0666))==-1)

{

perror("msgget:"); exit(1);

}

s2.mtype=1;

sprintf(s2.mtext,"%s","Hi friends... My name is message2"); if(msgsnd(mfd2,&s2,1000,0)==-1)

{

perror("msgsnd"); exit(1);

}

if((mfd3=msgget(1000,IPC\_CREAT|0666))==-1)

{

perror("msgget:"); exit(1);

}

s3.mtype=1;

sprintf(s3.mtext,"%s","Hi friends... My name is message3"); if(msgsnd(mfd3,&s3,1000,0)==-1)

{

perror("msgsnd"); exit(1);

}

printf("Your message has been sent successfully...\n"); printf("Please visit another (receiver's) terminal...\n"); printf("Thank you. For using LINUX\n");

}

# Output:

student@KITS:~/oslab$ cc mqsender.c student@KITS:~/oslab$ ./a.out

Your message has been sent successfully... Please visit another (receiver's) terminal... Thank you. For using LINUX

student@KITS:~/oslab$

# Receiver.c

#include<stdio.h> #include<stdlib.h> #include<sys/ipc.h> #include<sys/msg.h> #include<sys/types.h> #define SIZE 40

void main()

{

int mfd,mfd2,mfd3; struct

{

long mtype; char mtext[6];

}s1,s2,s3; if((mfd=msgget(1000,0))==-1)

{

perror("msgget"); exit(1);

}

if(msgrcv(mfd,&s1,SIZE,0,IPC\_NOWAIT|MSG\_NOERROR)==-1)

{

perror("msgrcv"); exit(1);

}

printf("Message from client is :%s\n",s1.mtext); if((mfd2=msgget(1000,0))==-1)

{

perror("msgget"); exit(1);

}

if(msgrcv(mfd2,&s2,SIZE,0,IPC\_NOWAIT|MSG\_NOERROR)==-1)

{

perror("msgrcv"); exit(1);

}

printf("Message from client is :%s\n",s2.mtext); if((mfd3=msgget(1000,0))==-1)

{

perror("msgget");

exit(1);

}

if(msgrcv(mfd3,&s3,SIZE,0,IPC\_NOWAIT|MSG\_NOERROR)==-1)

{

perror("msgrcv"); exit(1);

}

printf("Message from sender is :%s\n",s3.mtext);

}

# Output:

student@KITS:~/oslab$ cc mqclient.c student@KITS:~/oslab$ ./a.out

Message from client is :Hi friends... My name is message1 Message from client is :Hi friends... My name is message2 Message from client is :Hi friends... My name is message3 student@KITS:~/oslab$

# Shared Memory:

**DESCRIPTION:**

[Inter Process Communication](https://www.geeksforgeeks.org/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.

# PROGRAM:

shwriter.c

#include <stdio.h> #include <sys/ipc.h> #include <sys/shm.h> #include <sys/stat.h> int main ( )

{

int segment\_id; char bogus;

char\* shared\_memory; struct shmid\_ds shmbuffer; int segment\_size;

const int shared\_segment\_size = 0x6400;

/\* Allocate a shared memory segment. \*/

segment\_id = shmget (IPC\_PRIVATE, shared\_segment\_size, IPC\_CREAT

| IPC\_EXCL | S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IWGRP);

/\* Attach the shared memory segment. \*/

printf("Shared memory segment ID is %d\n", segment\_id); shared\_memory = (char\*) shmat (segment\_id, 0, 0);

printf ("shared memory attached at address %p\n", shared\_memory);

/\* Determine the segment's size. \*/

/\*

shmctl (segment\_id, IPC\_STAT, &shmbuffer); segment\_size = shmbuffer.shm\_segsz; printf ("segment size: %d\n", segment\_size);

\*/

/\* Write a string to the shared memory segment. \*/ sprintf (shared\_memory, "Hello, world.");

/\* Detach the shared memory segment. \*/ shmdt (shared\_memory);

printf("Wrote Hello World to the segment\n");

}

# PROGRAM:

shreader.c

#include <stdio.h> #include <sys/ipc.h> #include <sys/shm.h> #include <sys/stat.h> int main ()

{

int segment\_id; char bogus;

char\* shared\_memory; struct shmid\_ds shmbuffer;

int segment\_size;

const int shared\_segment\_size = 0x6400; printf("Enter the shared memory id: "); scanf("%d", &segment\_id);

/\* Reattach the shared memory segment, at a different address. \*/ shared\_memory = (char\*) shmat (segment\_id, (void\*) 0x5000000, 0); printf ("shared memory reattached at address %p\n", shared\_memory);

/\* Print out the string from shared memory. \*/

printf ("The contents of the shared memory is:\n%s\n", shared\_memory);

/\* Detach the shared memory segment. \*/ shmdt (shared\_memory);

return 0;

}

# OUTPUT:

**Terminal-I**

student@KITS:~/oslab$ cc shwriter.c student@KITS:~/oslab$ ./a.out Shared memory segment ID is 3047442

shared memory attached at address 0xb7f2a000 Wrote Hello World to the segment student@KITS:~/oslab$

# Terminal-II

student@KITS:~/oslab$ cc shreader.c student@KITS:~/oslab$ ./a.out

Enter the shared memory id: 3047442

shared memory reattached at address 0x5000000 The contents of the shared memory is:

Hello, world. student@KITS:~/oslab$