**LAB No.: 6 Date:**

**INTERPROCESS COMMUNICATION**

# Objectives:

In this lab, student will be able to:

* 1. Implement IPC using Shared Memory.

1. Start the program.

2. Declare the variables.

3. Read the choice.

4. Create a piping processing using IPC.

5. Assign the variable lengths

6. “strcpy” the message lengths.

7. To join the operation using IPC .

8. Stop the program

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). As a result, ordinary pipes are unidirectional, allowing only one-

way communication. If two-way communication is required, two pipes must be used, with

each pipe sending data in a different direction.

## PROGRAM: ( PIPE PROCESSING)

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#define MSG\_LEN 64

int main()

{

int result;

int fd[2];

char message[MSG\_LEN];

char recvd\_msg[MSG\_LE];

result = pipe (fd);

//Creating a pipe//fd[0] is for reading and fd[1] is for writing

if (result < 0)

{

perror("pipe ");

exit(1);

}

strncpy(message,"Linux World!! ",MSG\_LEN); result=write(fd[1],message,strlen(message));

if (result< 0)

{

perror("write");

exit(2);

}

strncpy(message,"Understanding ",MSG\_LEN); result=write(fd[1],message,strlen(message));

if (result < 0)

{

perror("write");

exit(2);

}

strncpy(message,"Concepts of ",MSG\_LEN); result=write(fd[1],message,strlen(message));

if (result < 0)

{

perror("write");

exit(2);

}

strncpy(message,"Piping ", MSG\_LEN); result=write(fd[1],message,strlen(message));

if (result < 0)

{

perror("write");

exit(2);

}

result=read(fd[0],recvd\_msg,MSG\_LEN);

if (result < 0)

{

perror("read");

exit(3);

}

printf("%s\n",recvd\_msg); return 0;

}

## a) FIFO

**Program:**

#include <stdio.h> #include <stdlib.h> #include <sys/stat.h> #include <unistd.h>

#include <linux/stat.h>

#define FIFO\_FILE "MYFIFO"

int main(void)

{

FILE \*fp;

char readbuf[80];

/\* Create the FIFO if it does not exist \*/ umask(0);

mknod(FIFO\_FILE, S\_IFIFO|0666, 0);

while(1)

{

fp = fopen(FIFO\_FILE, "r"); fgets(readbuf, 80, fp);

printf("Received string: %s\n", readbuf); fclose(fp);

}

return(0);

}

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

#define FIFO\_FILE "MYFIFO"

int main(int argc, char \*argv[])

{

FILE \*fp;

if ( argc != 2 ) {

printf("USAGE: fifoclient [string]\n"); exit(1);

}

if((fp = fopen(FIFO\_FILE, "w")) == NULL)

{ perror("fopen");

exit(1);

}

fputs(argv[1], fp);

fclose(fp); return(0);

}

C Program for Message Queue (Writer Process)

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

// structure for message queue struct mesg\_buffer {

long msg\_type; char msg\_text[100];

} message;

int main()

{

key\_t key; int msgid;

// ftok to generate unique key key = ftok("progfile", 65);

// msgget creates a message queue

// and returns identifier

msgid = msgget(key, 0666 | IPC\_CREAT); message.mesg\_type = 1;

printf("Write Data : "); gets(message.mesg\_text);

// msgsnd to send message

msgsnd(msgid, &message, sizeof(message), 0);

// display the message

printf("Data send is : %s \n", message.mesg\_text);

return 0;

}

C Program for Message Queue (Reader Process) #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;

// ftok to generate unique key key = ftok("progfile", 65);

// msgget creates a message queue and returns identifier msgid = msgget(key, 0666 | IPC\_CREAT);

// msgrcv to receive message

msgrcv(msgid, &message, sizeof(message), 1, 0);

// display the message printf("Data Received is : %s \n",

message.mesg\_text);

// to destroy the message queue msgctl(msgid, IPC\_RMID, NULL);

return 0;

}

C Program for Message Queue (Reader Process)

#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;

// ftok to generate unique key key = ftok("progfile", 65);

// msgget creates a message queue

// and returns identifier

msgid = msgget(key, 0666 | IPC\_CREAT);

// msgrcv to receive message

msgrcv(msgid, &message, sizeof(message), 1, 0);

// display the message printf("Data Received is : %s \n",

message.mesg\_text);

// to destroy the message queue msgctl(msgid, IPC\_RMID, NULL);

return 0;

}