**Operating System**

**Lab Report 5**

**Hafiz Ahmad**

**19l-1316**

Inter Process Communication using Pipes

**INTRODUCTION:**

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 oneway communication. If two-way communication is required, two pipes must be used, with each pipe sending data in a different direction. ¬ A pipe has a read end and a write end. ¬ Data written to the write end of a pipe can be read from the read end of the pipe. IPC is a very common mechanism in Linux and Pipe maybe one of the most widely used IPC methods. When you type **cat foo | grep bar** , you create a pipe to connect stdout of cat to stdin of grep . A pipe, as its name states, can be understood as a channel with two ends.

**OBJECTIVES:**

• Learn and Understand InterProcess Communication using implementation of Pipes

**Application:**

Pipe is a technique used for inter process communication. A pipe is a mechanism by which the output of one process is directed into the input of another process. Thus it provides one way flow of data between two related processes. Interprocess communication (IPC) is used for programs to communicate data to each other and to synchronize their activities. Semaphores, shared memory, and internal message queues are common methods of interprocess communication. A limitation of pipes for interprocess communication is that the processes using pipes must have a common parent process (that is, share a common open or initiation process and exist as the result of a fork system call from a parent process). A pipe is fixed in size and is usually at least 4,096 bytes. We use pipes **to transform strings, currency amounts, dates, and other data for display**. Pipes are simple functions to use in template expressions to accept an input value and return a transformed value. Pipes are useful because you can use them throughout your application, while only declaring each pipe once.

**Issues:**

No issue found regarding this lab.

**Conclusion:**

In this lab we learn and Understand Inter Process Communication using implementation of Pipes and we are able to communicate between process using pipes. Pipe is used **to combine two or more commands**, and in this, the output of one command acts as input to another command, and this command's output may act as input to the next command and so on. It can also be visualized as a temporary connection between two or more commands/ programs/ processes.

**Task 2:**

|  |  |
| --- | --- |
| #include<stdlib.h>  #include<iostream>  #include<stdio.h>  #include<unistd.h>  #include<sys/types.h>  #include<string.h>  int main(int argc, char\* arg[])  {  int p1[2];  pid\_t proc;  char s1[100],s2,s3[100];  pipe(p1);  proc=fork();  if(proc==0)  {  int j=0;  FILE \*fil;  fil=fopen(arg[1],"r");  do{  s2=fgetc(fil);  printf("%c",s2); | s1[j]=s2;  j++;  }while(s2 != EOF);  close(p1[0]);  write(p1[1],s1,j-1);  fclose(fil);  }  else  {  FILE \*cpf;  cpf= fopen(arg[2],"w");  close(p1[1]);  read(p1[0],s3,sizeof(s3));  fputs(s3,cpf);  fclose(cpf);  printf("%s",s3);  }  return 0;  } |