IPC Technique PIPES

Page No.
Date: / /20

→ A Pipe is a technique used for "Interprocess communication".

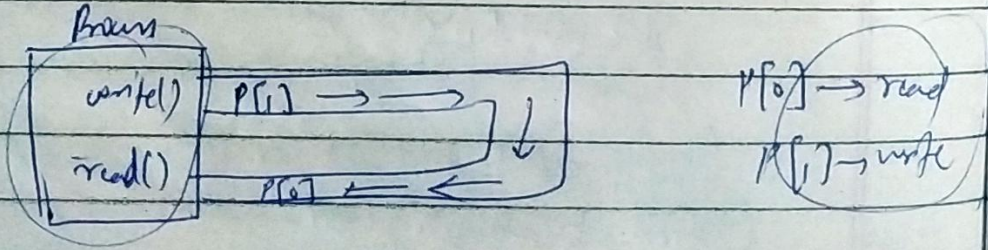
→ A Pipe is a mechanism by which the output of one process is directed into the input of another process.

→ Thus, it provides one way flow of data b/w two processes.

Suppose two processes, Process A & Process B, need to communicate. So, For [Comm. from A to B] below should happen:-

→ Process A, should keep its write end open and close the read end of the pipe.

→ Process B should keep its read end open and close write end. When a pipe is created, it is given a fixed size in bytes.



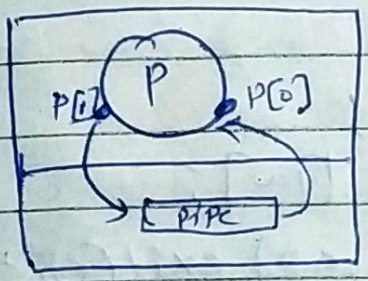
NDPC
Page No.:
Date : / /20

Syntax :- int pipe (int fds[2]);
(in C lang)

Parameters :-
→ fd[0] → fd (file descriptor) for read end of pipe.
→ fd[1] → fd (filedescriptor) for write end of pipe

Returns :- → 0 (on success)
→ -1 (on error)

1



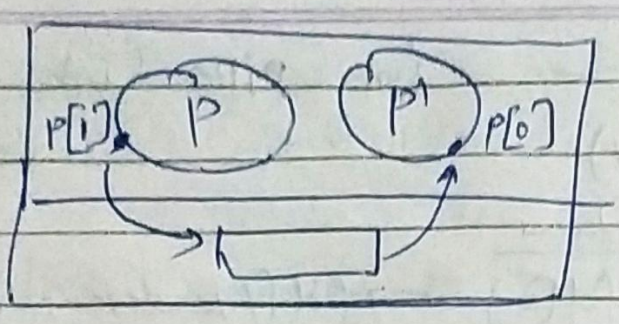

```
#include <unistd.h>
#include <stdio.h>
#define MSGSIZE 16

char* msg1 = "hello, world #1";
char* msg2 = "hello, world #2";
char* msg3 = "hello, world #3";

int main()
{ char buff[MSGSIZE];
  int p[2], i;
  if (pipe(p) < 0) exit(1);
  /* write pipe */
  write (p[1], msg1, MSGSIZE)
  write (p[1], msg2, MSGSIZE)
  write (p[1], msg3, MSGSIZE)
  /* read pipe */
  for (i=0; i<3; i++)
  {
    read (p[0], buff, MSGSIZE)
    printf ("%s\n", buff)
  }
  return 0;
}
```

output
hello, world #1
hello, world #2
hello, world #3

NDPC
Page No.:
Date : / /20

(2)

one-way communication



→ close P[0], for Parent Process(P)
→ close P[1], for Child Process(P')

```
#include<iostream>
#include<unistd.h>
using namespace std;
int main()
{
    int P[2];
    char buf[100];
    if(Pipe(P) < 0) exit(1);
    int c = fork();
    if(c>0)
    {
        close(P[0]);
        cin.getline(buf,100); // reading but it work only 5 chars
        write(P[1],buf,5);
    }
    else
    {
        close(P[1]);
        read(P[0],buf,5);
        write(1,buf,5); // work on screen which
    }
}
```

Input:- abcdefg
abcde
output on screen

Note:

NDPC
Page No.:
Date: / /20

fgets() & fputs() # Functions

(1) char *fgets(char *str, int n, FILE *stream)

where (a) str → This is the pointer to an array of chars where string read is stored.

(b) n → Max. no. of characters to be read (including final null-character). Usually the length of the array passed as str is used.

(c) stream → This is the pointer to a FILE object that identifies the stream where characters are read from.

Example

```
int main() {
    FILE *fp;
    char str[60];

    fp = fopen("file.txt", "r"); //read file
    if (fp == NULL) {
        perror("Error opening file");
        return -1;
    }
    if (fgets(str, 60, fp) != NULL) {
        Content str
    }
    fclose(fp);
    return 0;
}
```

NDPC
Page No.
Date : / / 20

(2) int puts (const char *str, FILE *stream)

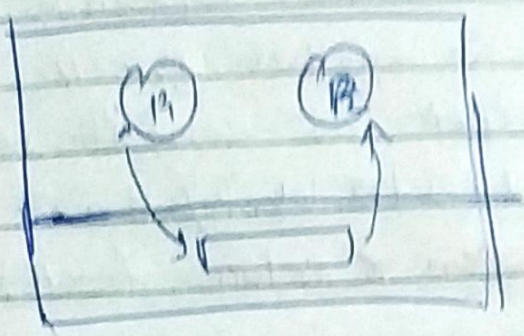
Parameters :-

(a) str → This is an array containing the null-terminated sequence of characters to be written.

(b) stream → This is the pointer to a FILE object that identifies the stream where the string is to be written.

fgets() ⇒ returns, a pointer to the string buffer if successful. else return NULL.

③ IPC - one way Communication (P1), (P2)



P1.cpp

```
int main()
{ int A[2];
  pipe(A);
  char buf1[100];
  int pid = fork();
  if(pid > 0)
  { close(A[0]);
    while(1) {
      cin >> buf1;
      write(A[1], buf1, 10);
    }
  }
  else
  { close(A[1]);
    dup2(A[0], 0);
    execvp("./P2", NULL);
  }
}
```

P2.cpp

```
int main()
{
  char buf2[100];
  while(1) {
    read(0, buf2, 10);
    char *str = buf2;
    while(*str) {
      *str = toupper(*str);
      str++;
    }
    cout << "P2: " << buf2 << endl;
    fflush(stdout);
  }
}
```

input abcdef

output P2: abcdef

Popen & Pclose functions

→ Since common operation is to create a pipe to another process, to either read its output or send it input; the standard I/O library has provided the popen & pclose functions.

→ These two functions handle all the dirty work that we have doing ourselves:—

↳ creating a pipe, forking a child, closing the unused ends of the pipe, executing a shell to run the command, & waiting for the command to terminate.

(i) `#include <stdio.h>`

(ii) `FILE *popen(const char *cmdstring, const char *type)`

↳ Returns: file pointer if OK, NULL on error

(iii) `int pclose(FILE *fp)`

↳ Returns: termination status of cmdstring or -1 on error.

NDPC
Page No.:
Date : / / 20

→ The function popen does a fork and exec to execute the cmdstring, & returns a standard I/O file pointer.

- (i) If type is "r" the file pointer is connected to the standard output of cmdstring.
- (ii) If type is "w" the file pointer is connected to the standard input of cmdstring.

→ The function pclose, closes the standard I/O stream, waits for the command to terminate, & returns the termination status of the shell.

FIFO # (IPC)

Page No.. NDPC
Date : / / 20

→ Known as "Named pipe".

→ Creating a FIFO file :—

int mkfifo (const char * pathname, mode_t mode);

→ Here, mode specifies the FIFO's permissions

using FIFO : As named pipe (FIFO) is a kind of file, we can use all the system calls associated with it i.e open, read, write, close.

Example Program —

There are two programs that use the same FIFO, Program 1 writes first, then reads. The programs read first, then writes. They both keep doing it until terminated.

Libraries

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

Page No.:
Date: / /20

Program 1 (writefirst) :-

```
#include<stdio.h>
int main()
{
    int fd;
    char *myfifo = "/tmp/myfifo";
    mkfifo(myfifo, 0666);
    char arr1[80], arr2[80];
    while(1)
    {
        // open FIFO for write only
        fd = open(myfifo, O_WRONLY);
        // take input from user.
        fgets(arr2, 80, stdin);
        // write input on FIFO
        write(fd, arr2, strlen(arr2)+1);
        close(fd);

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

NDPC
Page No.:
Date: / /20

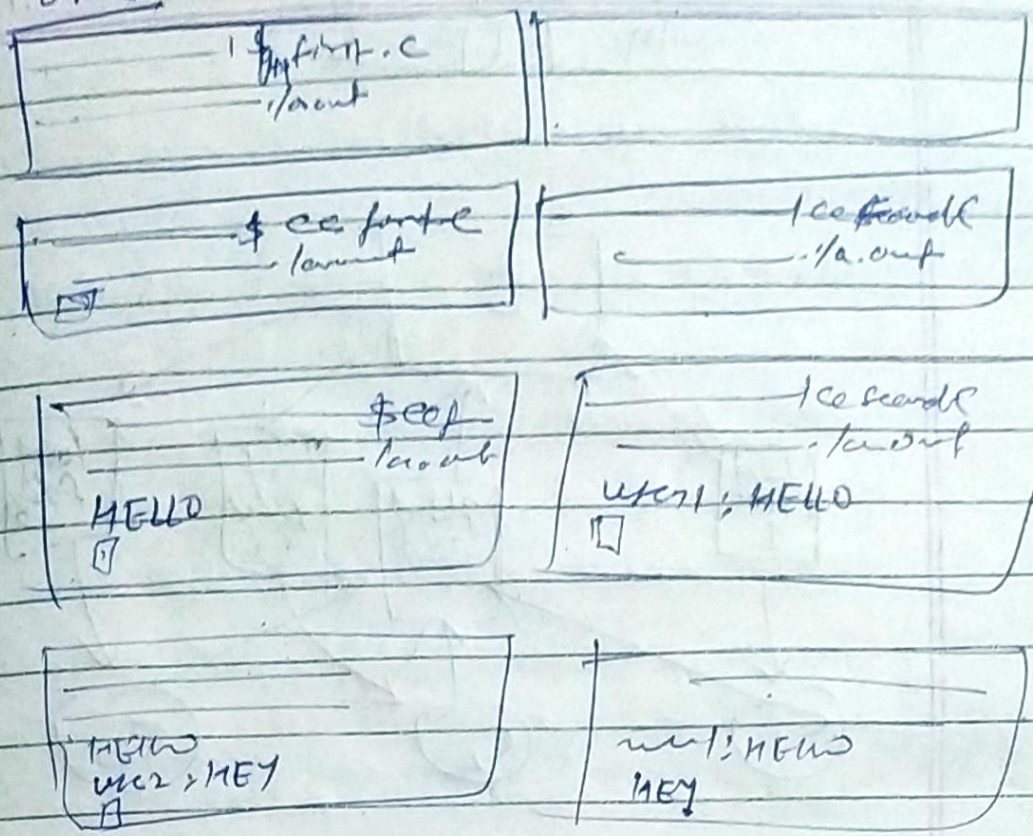
Program 2 (Read first)

```
int main()
{
  int fd1;
  char * myfifo = "/tmp/myfifo";
  MKfifo(myfifo, 0666); // create FIFO
  char str1[80], str2[80];
  while(1)
  {
    // first open in readonly & read
    fd1 = open(myfifo, O_RDONLY);
    read(fd1, str1, 80);
    printf("User1: %s\n", str1);
    fd1 = open(myfifo, O_WRONLY);
    fgets
    // Now take input from user (in write mode)
    fd1 = open(myfifo, O_WRONLY);
    fgets(str2, 80, stdin);
    write(fd1, str2, strlen(str2)+1);
    close(fd1);
  }
  return 0;
}
```

NDPC
Page No.:
Date : 04/2/20

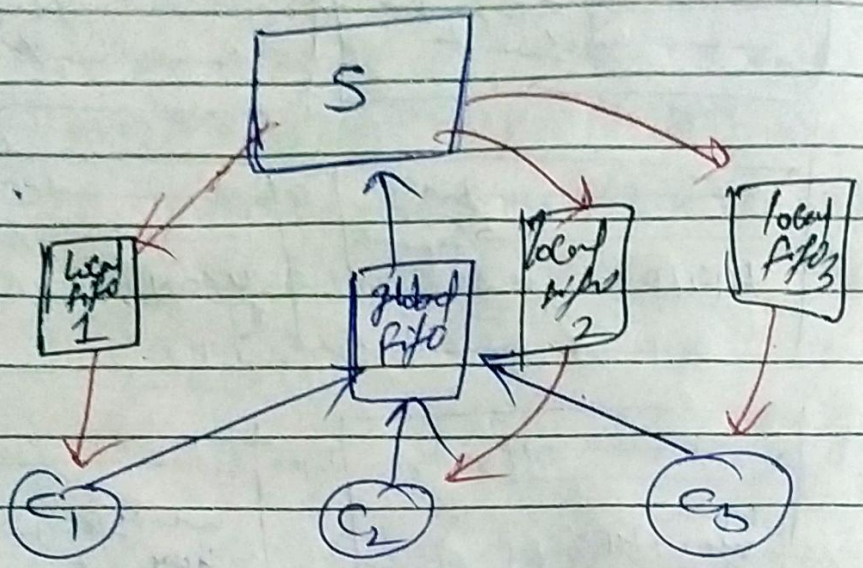
Lab

user1



NDPC
Page No.:
Date : / /20

client - server with FIFO



C1 → write
C2 → read
C3 → write

globalFifo
localFifo1
localFifo2