CSE 6 'D'

1DA18CS173

UJVAL DR

UNIX PROGRAMMING - 18CS63

Assignment - 5

1. Explain pipes with neat diagram.

A: A pipe is a connection between two processes, such that the standard output from one process becomes the standard input of the other process. In Unix operating system, pipes are useful for communication between related process (Inter process Communication)

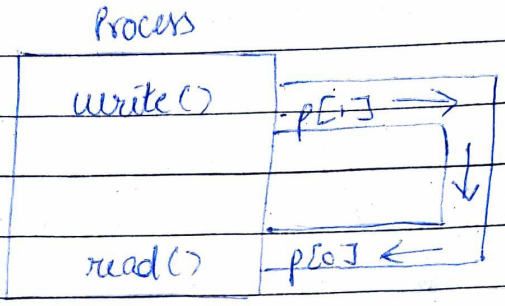
- Pipe is one way communication only i.e, we can use a pipe such that one process write to the pipe and other process reads from pipe. It opens a pipe, which is an area of main memory is treated as a virtual file.

- The pipes can be used by the creating process, as well as all its child processes, for reading and writing one process can write to this "virtual file" or pipe and another related process can read from it.

- If a process tries to read before something is written to the pipe, the process is suspended until something is written.

UJVAL DR
1DA18CS173

- The pipe system call finds two available positions in the process's open file table and allocates them for the read and write ends of the pipe.



Program :

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

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

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

    if (pipe(p)<0)
        exit(1);
```

UJVAL DR
1DA18CS173

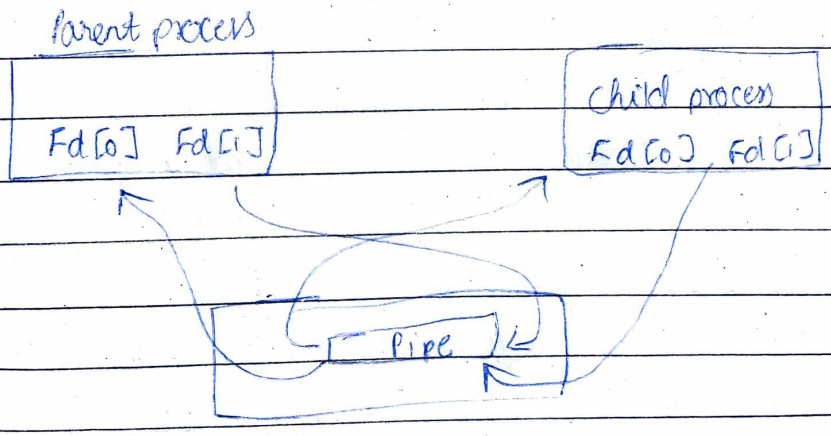
```
write (p[1], msg1, MSGSIZE);
write (p[1], msg2, MSGSIZE);
write (p[1], msg3, MSGSIZE);

for (i=0; i<3; i++) {

    read (p[0], inbuf, MSGSIZE);
    printf ("%.s\n", inbuf);
}

    return 0;
}
```

When we use fork in any process, file descriptors remain open across child process and also parent process. If we call fork after creating a pipe, then the parent and child can communicate via the pipe.



UJVAL DR
IDA18CS173

(2) Explain Popen and pclose Files.

A: Since a common operation is to create a pipe into another process, to either a read its output or send it input, the standard I/O library has historicaly provided the popen and pclose files. functions. These two functions handle all the dirty work that we been doing ourselves: Creating a pipe, forking a child, closing the unused ends of the pipe, executing a shell to run the command, and waiting for the command to terminate.

```
#include <stdio.h>

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

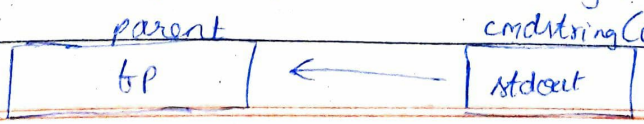
Returns: file pointer if OK, NULL on Error.

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

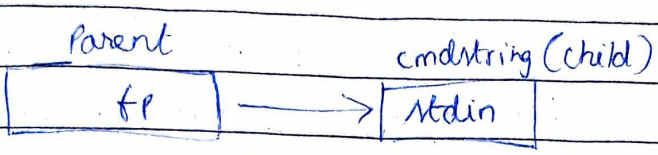
Returns: Termination status of cmdstring, or 1 on error.

The function popen closes a fork and exec to execute the cmdstring, and returns a standard I/O file pointer. If type is "r", the file pointer is connected to the standard out of cmd string.

Result of fp = popen (cmdstring, "r")



- if type is "w", the file pointer is connected to the standard input of cmdstring, as shown:



(3) Explain briefly FIFO in IPC.

A: FIFO's are sometimes called pipes. Pipes can be used only between related process when a common ancestor has created the pipe.

```
#include <sys/stat.h>
int mkfifo (const char *pathname, mode_t mode);
```

Returns: 0 if OK; -1 on error

Once we have used mkfifo to create a FIFO, we open it using open. When we open a FIFO, the nonblocking flag (O_NONBLOCK) affects what happens.

- In the normal case (O_NONBLOCK not specified), an open for read-only blocks until someone other process opens the FIFO for writing. Similarly, an open for write-only blocks until some other process open the FIFO for reading.

- If the O_NONBLOCK is specified, an open for read-only returns immediately. But an open for write-only returns -1 with errno set to ENXIO if no process has the FIFO open for reading.

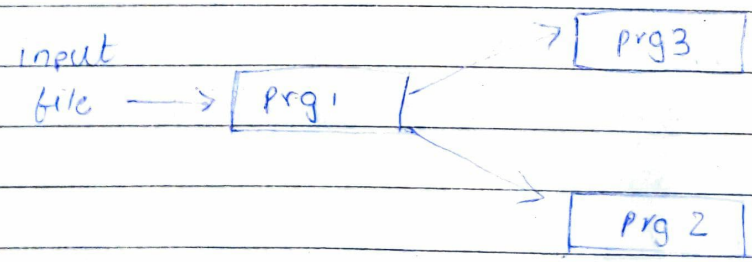
There are two uses for FIFO's

* FIFO's are used by shell commands to pass data from one shell pipeline to another without creating intermediate temporary files.

* FIFO's are used as rendezvous point in client-server applications to pass data between the clients and the servers.

Example using FIFO's to duplicate output streams

FIFO's can be used to duplicate an output stream in a series of shell commands. This prevents writing the data to an intermediate disk file. Consider a procedure that needs to process a filtered input stream twice.



procedure that processes a filtered input stream twice

With a FIFO and the UNIX program, we can accomplish this procedure without using a temporary file.

UJVAL DR
IDA18CS173

```
mkfifo fifo1
prg3 < fifo1 &
prg1 < infile1 tee fifo1 | prog2
```

We create the FIFO and then start prog3 in the background reading from the FIFO. We then start prg1 and use tee to send its input to both the FIFO and prg2.

(4) Explain briefly how Socket Communication takes place between client and Server with neat diagram.

A:

Client process: This process, which typically makes a request for information. After getting the response this process may terminate or may do some other processing.

Server process: This is the process which takes a request from the clients. After getting a request from the client, this process will perform the required preprocessing, gather the requested information, and send it to the request or client. Once done, it becomes ready to serve another client Server processes are always alert and ready to serve incoming requests.

UJVAL DR
IDA18CS173

2-Tier and 3-Tier architectures:

There are two types of client-server architectures:

* 2-Tier architecture: In this architecture, the client directly interacts with the server. This type of architecture may have some security holes and performance problems. Internet Explorer and Web server work on two tier architecture. Here security problems are resolved using Secure Socket Layer (SSL)

* 3-Tier architecture:

In this architecture one or more software sits in between the client and server. The middle software is called "middleware". Middlewares are used to perform all the security checks and load balancing in case of heavy load. A middleware takes all requests from the client and after performing the required authentication, it passes that request to the server. Then the server does the required processing and sends the response back to the middleware and finally the middleware passes this response back to the client. If you want to implement a 3 tier architecture, then you can keep any middleware like web logic or websphere

UJVAL DR
1DA18CS173

Types of Server:

There are two types of servers you can have-

Iterative server - This is the simplest form of server where a server process serves one client and after completing the first request, it takes request from another client. Meanwhile another client keeps waiting.

Concurrent servers - This type of server runs multiple concurrent processes to serve many requests at a time because one process may take longer and another client cannot wait for so long. The simplest way to write a concurrent server under Unix is to fork a child process to handle each client separately.

Making a client:

The system calls for establishing a connection are somewhat different for the client and the server but both involve the basic construct of a socket. Both the process establish their own sockets.

The steps involving in establishing a socket on the client side are as follows:

* Create a socket with socket() system call.
* Connect the socket to the address of the server using connect() system call.

UJVAL DR
1DA18CS173

Making a server:

steps:

* Create a socket with the socket() system call.
* Bind the socket to an address using the bind() system call. For a server socket on the Internet. An address consists of a port number on the host machine.
* Listen for a connections with the listen() system call
* Accept a connection with the accept() system call. This call typically blocks the connection until a client connects with the server.
* Send and receive data using the read() and write() system calls.