ASSIGNMENT NO. 3

Title : Write a program to stimulate interprocess communication mechanism using pipe and redirection.

Objective : To understand pipe operations mechanism and redirection.

Software Requirement : Ubuntu, JDK, text editor

Theory :

• What is pipe ?

↳ Pipe is a communication between two or more relatable or interrelated processes, it can be communication between child and parent process. Pipe mechanism is to filling water with pipe into container to filling process is nothing but writing into the file and reading process is nothing but receiving data from the file.

• Communication :

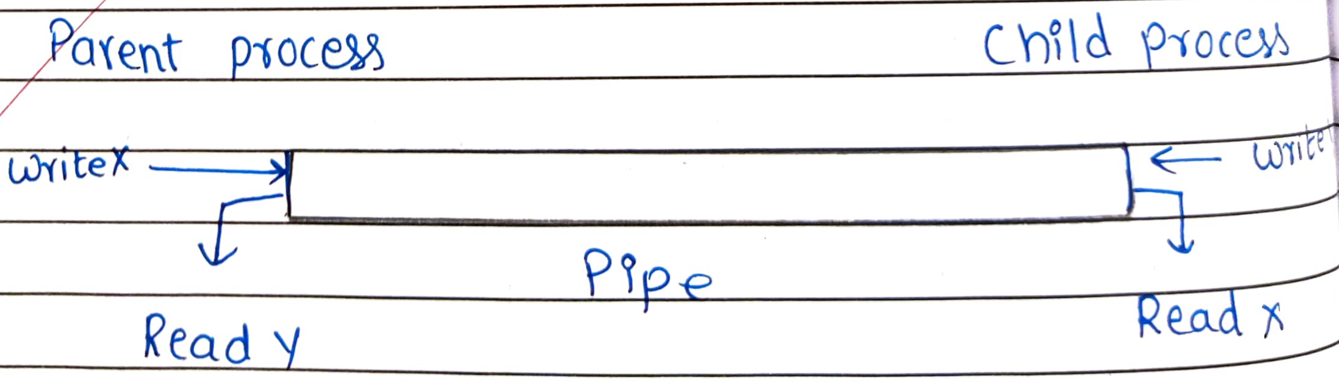
↳ Communication can be multilevel such as communication between parent and child and grand - child.

↳ By one process writting into the pipe and other accessing from the pipe to acheive pipe system calls. 12 files need to be created one to write into file and another to read from file.

```
# include < unistd.h >
int pipe (int pipedes [2]) ;
```

This system call would create a pipe from one way communication i.e., it creates two descriptors first read from the pipe and second write into pipe, this call return 0 & 1 in case of success & failure.

• Two way communication using pipe
↳ Pipe communication is viewed as only one-way communication i.e, either the parent process writes and child process reads or vice versa but not both. When both parent and child wants to read and write from the pipe simultaneous the two way communication using pipe is done, they are required to establish two way communication.



• **Step :-** Two way communication

1] Create a pipe1 for parent to write and the child process read.

2] Create a pipe2 for child process to write and parent process read.

3] Remove unwanted ends from both pipes, from sides.

4] Parent process to write a message and child process to read and display on the screen.

5] Child process to write a message and parent process to read and display on the screen.

• **dup2()**

↳ The dup2() system call is similar to dup() but the basic difference between them is that instead of using the lowest-numbered unused file descriptor, it uses the descriptor number specified by the user.

Syntax :

int dup2 (int oldfd, int newfd);

oldfd : old file descriptor

newfd new file descriptor which is used by dup2() to create a copy.

As in dup2() , in place of newfd any file descriptor can be put. Below is a C implementation in which the file descriptor of standard output (stdout) is used. This will lead all the printf () statements to be written in the file referred by the old file descriptor.

Conclusion :

Thus, we have implemented two way communications using pipe function.

ASSIGNMENT NO. 4

Title : Write a program using pthreads to demonstrate the reader - writer synchronization problem. Implemented appropriate synchronization. Show the different results with and without synchronisation.

Objective : To study the working and processing of reader - writer synchronisation problem.

Outcome : To implement reader- writer problem using pthreads.

Software Requirement : gcc / JDK or eclipse

Theory :

• Reader - writer's Problem

↳ If one of the user tries editing the file, no other person should be reading or writing at the same time, otherwise changes is reading will not be visible.

↳ However, if some person is reading the file, then other may ~~record~~ read it at the same time but could not write. In OS, we call this situation as reader - writter problem.

• ~~writing process~~
↳ writer requests the entry to critical section

```
do
{
    // writer requests for critical section
    wait (wrt);
    // performs the write
    // leaves the critical section
    signal (wrt);
} while (true);
```

• Reader's process

```
do
{
    wait (mutex);
    read cnt ++;
    if (read int == 1)
    wait (wrt);
    signal (mutex);
    wait (mutex);
    readent --;
    if (readcnt == 0)
        signal (wrt);
    signal (mutex);
} while (true);
```

• Readers - Writers problem using pthreads
↳ There is a shared resources that is ac
by multiple processes i.e., readers and from
the shared resources simultaneously, but

one writes can write to the shared resource at a time. When a writer is writing data to the resource no other process can access the resource. A writer cannot write to the resource, if there are many waiting writers.

The reader - writer problem using a monitor can be implemented using pthreads. The posix threads or (pthread) libraries are a standard provides the following synchronisation mechanism.

1. Mutex (pthread - mutex) : Mutal exclusion lock : Block access to variable by other threads. This enforces exclusion access by thread to a variable a set of variable.

2. Condition variables (pthread - cont) : The condition variable mechanism allows threads to suspend execution and relinquish the processor until some condition is true.

• Conditions :

1. Readers can access database only when there are no writers.
2. Writers can access database only when there are no readers or writters.
3. Only one thread can manipulate the state variables at a time.

• Basic structure of solution
Reader ()
wait until no writers access database
Check - out wake up a waiting writer.

Writer ()
wait until no active reader - or - writer
access database. check out wake up
waiting readers or writes.

↳ The reader - writer problem is used to manage synchronization so that there are no problems with the object data.
For example : If two readers access the object at the same time. There is no problem. However, if two writers or a writer and reader access the object at the same time those may be problems.

Conclusion :
Successfully implemented the Reader-writer problem using pthreads.