Lab 10:

Inter-process Communication

10.1 Pipes

The capacity to communicate is essential for processes that cooperate to solve a problem. The simplest UNIX interprocess communication mechanism is the pipe, which is represented by a special file. The pipe function creates a communication buffer that the caller can access through the file descriptors fildes[0] and fildes[1]. The data written to fildes[1] can be read from fildes[0] on a first-in-first-out basis.

```
SYNOPSIS

  #include <unistd.h>

  int pipe(int fildes[2]);
```

If successful, pipe returns 0. If unsuccessful, pipe returns –1 and sets errno.
A pipe has no external or permanent name, so a program can access it only through its two descriptors. For this reason, a pipe can be used only by the process that created it and by descendants that inherit the descriptors on fork. The pipe function described here creates a traditional unidirectional communication buffer. The POSIX standard does not specify what happens if a process tries to write to fildes[0] or read from fildes[1].

When a process calls read on a pipe, the read returns immediately if the pipe is not empty. If the pipe is empty, the read blocks until something is written to the pipe, as long as some process has the pipe open for writing. On the other hand, if no process has the pipe open for writing, a read from an empty pipe returns 0, indicating an end-of-file condition. (This description assumes that access to the pipe uses blocking I/O.)

10.2 FIFOs

Pipes are temporary in the sense that they disappear when no process has them open. POSIX represents FIFOs or named pipes by special files that persist even after all processes have closed them. A FIFO has a name and permissions just like an ordinary file and appears in the directory listing given by `ls`. Any process with the appropriate permissions can access a FIFO. Create a FIFO by executing the `mkfifo` command from a shell or by calling the `mkfifo` function from a program.

The `mkfifo` function creates a new FIFO special file corresponding to the pathname specified by `path`. The `mode` argument specifies the permissions for the newly created FIFO.

```
SYNOPSIS

   #include <sys/stat.h>

   int mkfifo(const char *path, mode_t mode);
```

If successful, `mkfifo` returns 0. If unsuccessful, `mkfifo` returns –1 and sets `errno`. A return value of –1 means that the FIFO was not created.

10.3 Lab Tasks

Task 1:

A program in which a child writes a string to a pipe and the parent reads the string.

Task 2:

Write a program that creates a process fan. Parent process writes to the pipe and all the child processes read the message from pipe and display it on stdout.

Task 3:

Chatting between two process using FIFO

Write a Chatting application in which two processes can communicate using FIFO. Your program should satisfy the following specifications.
The program should take the name of FIFO, will create the FIFO (if not created yet) and should open it for reading and writing. Program should take input from standard input and write it to FIFO and should read from FIFO and write to standard output in another process. Both reading and writing shall be done concurrently.