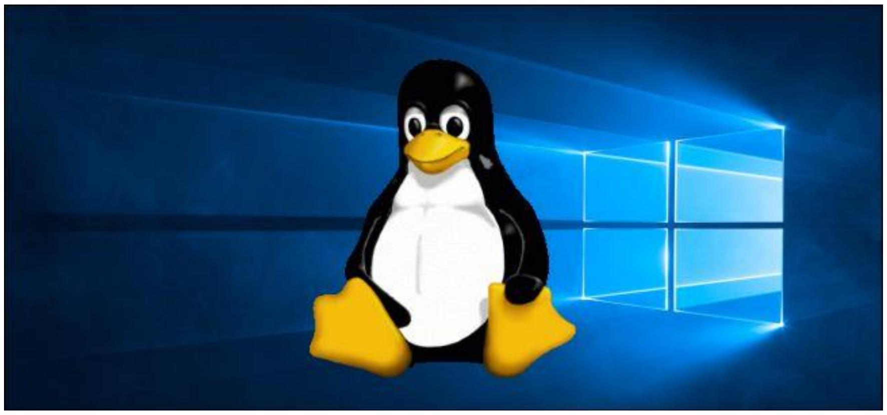
UNIVERSITY OF ENGINEERING AND TECHNOLOGY, TAXILA

FACULTY OF TELECOMMUNICATION AND INFORMATION ENGINEERING

SOFTWARE ENGINEERING DEPARTMENT



Operating Systems

Experiment 10

IPC (Inter Process Communication)

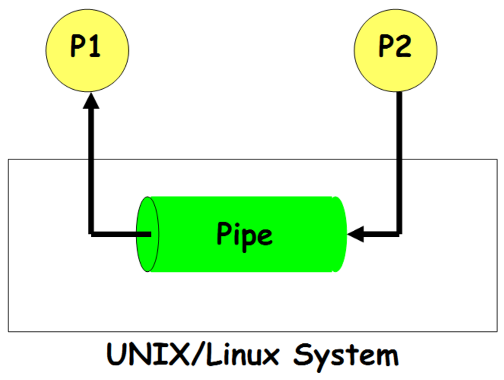
CLO 2. Use modern tools and languages.
CLO 3. Demonstrate an original solution of problem under discussion.
CLO 4. Work individually as well as in teams

IPC using PIPE:

IPC:

It is a mechanism for processes to communicate and to synchronize their actions.

Pipe:



- One of the mechanisms that allow related-processes on a system to communicate is the pipe. A pipe is a one-way mechanism that allows two related processes (i.e. one is an ancestor of the other) to send a byte stream from one of them to the other one.
- The system assures us of one thing: The order in which data is written to the pipe, is the same order as that in which data is read from the pipe. The system also assures that data won't get lost in the middle, unless one of the processes (the sender or the receiver) exits prematurely.
- A pipe is a method of connecting the standard output of one process to the standard input of another.
- This direct connection between programs allows them to operate simultaneously and permits data to be transferred between them continuously rather than having to pass it through temporary text files or through the display screen and having to wait for one program to be completed before the next program begins.

Command Line Use of PIPE:

Connect standard output of a command to standard input of another.

- Use the pipe operator |
- **Syntax:** cmd1 | cmd2 | … | cmdN

EXAMPLE 1:

The following example employs a pipe to combine the ls and the wc (i.e., word count) commands in order to show how many *filesystem objects* (i.e., files, directories and links) are in the current directory:

```
$ ls | wc -l
```

- ls lists each object, one per line, and this list is then piped to wc, which, when used with its -l option, counts the number of lines and writes the result to standard output.
- The output could be redirected to a file named, for instance, count.txt:

```
$ ls | wc -l > count.txt
```

i. The output redirection operator will create count.txt if it does not exist or overwrite it if it already exists. And the output will be redirected to it.

PIPE in C Programming:

Creating "pipelines" with the C programming language can be a bit more involved than our simple shell example.

The pipe () system call:

- To create a simple pipe with C, we make use of the pipe() system call.
- It takes a single argument, which is an array of two integers,
- If successful, the array will contain two new file descriptors to be used for the pipeline.
- After creating a pipe, the process typically initiates a new process (remember the child inherits open file descriptors).

#include <unistd.h>
Prototype: int pipe(int fd[2]);
Here is how to use this function:

```
int fd[2];
if (pipe(fd) < 0)
perror("Error");
```

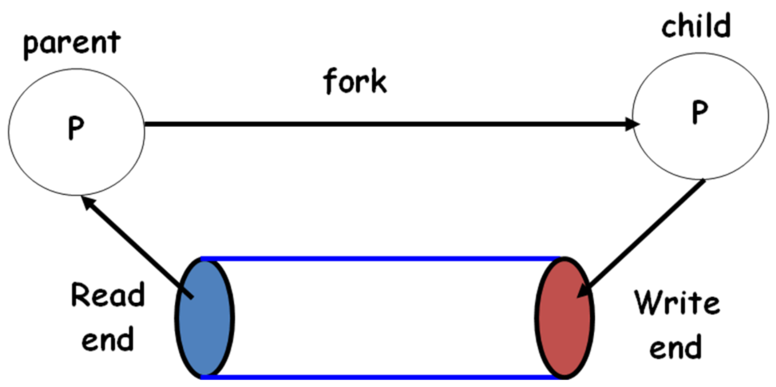
Returns:

- 0 on success
- -1 on error

NOTE:

- fd[0] is set up for reading
- fd[1] is set up for writing

The first integer in the array (element 0) is set up and opened for reading, while the second integer (element 1) is set up and opened for writing. Visually speaking, the output of *fd1* becomes the input for *fd0*.



IMPORTANT SYSTEM CALLS:

- **open:** Open or create a file
- **read:** Read from a pipe
- **write:** Write data to a pipe
- **close:** Close/destroy a pipe
- **pipe:** Create a pipe for IPC

Implementation of pipe() System Call

Example1:

/* Parent creates pipe, forks a child, child writes into pipe, and parent reads from pipe */

Program:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include <unistd.h>
main()
{
int pipefd[2], pid, n, rc, nr, status;
char *testString = "Hello, world!\n";
char buf[1024];
rc = pipe (pipefd);
if (rc < 0)
        {
perror("pipe");
}
pid = fork ();
if (pid < 0)
{
perror("fork");
}
if (pid == 0)
{       /* Child's Code */
close(pipefd[0]);// read is closed for Child, it will use pipe to Write using fd1.
write(pipefd[1], testString, strlen(testString));
close(pipefd[1]);

}
else            /* Parent's Code */
{
close(pipefd[1]);// write is closed for Parent, it will use pipe to Read using fd0
n = strlen(testString);
nr = read(pipefd[0], buf, n);
rc = write(1, buf, nr);
wait(&status);
printf("Good work child!\n");
}
return(0);
}
```

Example 2:
In the following example, it is necessary to create an unnamed pipe at the beginning of a parent process, and then create two child processes. One of these child processes should execute some shell command (for example, **"who")** and write the results of this command into the **upstream** end of the pipe. Another child process should execute some other shell command (for example, **"sort")** taking its input from the **downstream** end of the same pipe. Therefore, the complete effect of executing the two child processes can be expressed as the sequence of two shell commands.

who | sort

Make the following experiments:

1. Create **a separate subdirectory** for this lab work. Input and create executable file, corresponding to the following C program:

```
/* A program to experiment with an unnamed pipe */
#include <stdio.h>
#include <unistd.h>

main ()
{
int fd[2];  /*  Array of two descriptors for an unnamed  pipe */
int pid;     /*  Variable for a process identifier */

/*  A pipe should be created before any fork() */

if (pipe(fd) < 0)
      {perror ("PIPE CREATION ERROR");
       exit (1);
      }

pid = fork ();  /*  Parent:  creating the first child process */
if (pid == 0)   /*  The first  child process starts here */
      {
      dup2 (fd[0],0);  /* Standard input will be taken from the  downstream of the pipe  */
      close (fd[1]);     /*  Upstream end of the pipe is closed for this process (not used) */
      execlp ("sort", "sort", NULL);  /* Running  "sort" command  taking input from the
pipe */
      }
else   /* Here the parent process continues */
pid = fork ();  /*  Parent:  creating the second child process */
if (pid == 0)   /*  The second child process starts  here */
```

```
        {dup2 (fd[1],1); /* Standard output  will be put  to the upstream end of the pipe */
         close (fd[0]);    /* Downstream end is closed  for this process (not used) */
         execlp ("who", "who", NULL); /*  Running the command “who” which outputs to
    the pipe */
        }
    else /* Parent process closes for itself both ends of the pipe and waits for children to
    terminate */
        {
        close (fd[0]);
        close (fd[1]);
        wait (0);
        wait (0);
        }
    }
```

Example:3

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

main ()
{
int fd[2];  /*  Array of two descriptors for an unnamed  pipe */
int pid;     /*  Variable for a process identifier */

/*  A pipe should be created before any fork() */

if (pipe(fd) < 0)
{
perror ("PIPE CREATION ERROR");
exit (1);
}

pid = fork ();  /*  Parent:  creating the first child process */
if (pid == 0)   /*  The first  child process starts here */
{
dup2 (fd[0],0);  /* Standard input will be taken from the  downstream of the pipe  */
close (fd[1]);     /*  Upstream end of the pipe is closed for this process (not used) */
execlp ("sort", "sort", "-r", NULL);  /* Running  â€œsortâ€•  command  taking input from the
pipe */
}
else   /* Here the parent process continues */
pid = fork ();  /*  Parent:  creating the second child process */
if (pid == 0)   /*  The second child process starts  here */
{
dup2 (fd[1],1); /* Standard output  will be put  to the upstream end of the pipe */
close (fd[0]);    /* Downstream end is closed  for this process (not used) */
execlp ("ls", "ls", NULL); /*  Running the command â€œwhoâ€•   which outputs to the pipe */
}
else /* Parent process  closes for itself both ends of the pipe and waits for  children to terminate
*/
{
close (fd[0]);
close (fd[1]);
wait (0);
wait (0);
}
}
```

UNIVERSITY OF ENGINEERING AND TECHNOLOGY, TAXILA

FACULTY OF TELECOMMUNICATION AND INFORMATION ENGINEERING

SOFTWARE ENGINEERING DEPARTMENT

LAB TASK

Write a program of creation of a two-way pipe between two process (Child process sends a question to Parent process "where is GEC?" through one pipe, parent process reads this questions and send a reply "in gudlavalleru" through second pipe which the child process reads.)