GROUP-C

EXPERIMENT NO.1

Title –

Aim:- To study Inter process communication in Linux using Pipes.

Theory:-

A Pipe is a technique used for inter process 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 between two related processes.

Although pipe can be accessed like an ordinary file, the system actually manages it as FIFO queue. A pipe file is created using the pipe system call. A pipe has an input end and an output end. One can write into a pipe from input end and read from the output end. A pipe descriptor, has an array that stores two pointers, one pointer is for its input end and the other pointer is for its output end.

Suppose two processes, Process A and Process B, need to communicate. In such a case, it is important that the process which writes, closes its read end of the pipe and the process which reads, closes its write end of a pipe. Essentially, for a communication from Process A to Process B the following 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 its write end. When a pipe is created, it is given a fixed size in bytes.

When a process attempts to write into the pipe, the write request is immediately executed if the pipe is not full.

However, if pipe is full the process is blocked until the state of pipe changes. Similarly, a reading process is blocked, if it attempts to read more bytes that are currently in pipe, otherwise the reading process is executed. Only one process can access a pipe at a time.

Limitations :

As a channel of communication a pipe operates in one direction only.

Pipes cannot support broadcast i.e. sending message to multiple processes at the same time.

The read end of a pipe reads any way. It does not matter which process is connected to the write end of the pipe. Therefore, this is very insecure mode of communication.

Some plumbing (closing of ends) is required to create a properly directed pipe.

To create child process we use fork(). fork() returns :

<0 fail to create child (new) process

=0 for child process

>0 i.e process ID of the child process to the parent process. When >0 parent process will execute.

pipe() is used for passing information from one process to another. pipe() is unidirectional therefore, for two-way communication between processes, two pipes can be set up, one for each direction.

Inside Parent Process : We firstly close the reading end of first pipe (fd1[0]) then write the string though writing end of the pipe (fd1[1]). Now parent will wait until child process is finished. After the child process, parent will close the writing end of second pipe(fd2[1]) and read the string through reading end of pipe (fd2[0]).

Inside Child Process : Child reads the first string sent by parent process by closing the writing end of pipe (fd1[1]) and after reading concatenate both string and passes the string to parent process via fd2 pipe and will exit.

CODE:-

```
// C program to demonstrate use of fork() and pipe()
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<string.h>
#include<sys/wait.h>

int main()
{
	// We use two pipes
	// First pipe to send input string from parent
	// Second pipe to send concatenated string from child

	int fd1[2]; // Used to store two ends of first pipe
	int fd2[2]; // Used to store two ends of second pipe

	char fixed_str[] = "forgeeks.org";
	char input_str[100];
	pid_t p;

	if (pipe(fd1)==-1)
	{
```

```
        fprintf(stderr, "Pipe Failed" );
        return 1;
}
if (pipe(fd2)==-1)
{
        fprintf(stderr, "Pipe Failed" );
        return 1;
}

scanf("%s", input_str);
p = fork();

if (p < 0)
{
        fprintf(stderr, "fork Failed" );
        return 1;
}

// Parent process
else if (p > 0)
{
        char concat_str[100];

        close(fd1[0]); // Close reading end of first pipe

        // Write input string and close writing end of first
```

```
        // pipe.
        write(fd1[1], input_str, strlen(input_str)+1);
        close(fd1[1]);

        // Wait for child to send a string
        wait(NULL);

        close(fd2[1]); // Close writing end of second pipe

        // Read string from child, print it and close
        // reading end.
        read(fd2[0], concat_str, 100);
        printf("Concatenated string %s\n", concat_str);
        close(fd2[0]);
}

// child process
else
{
        close(fd1[1]); // Close writing end of first pipe

        // Read a string using first pipe
        char concat_str[100];
        read(fd1[0], concat_str, 100);

        // Concatenate a fixed string with it
```

```
            int k = strlen(concat_str);
            int i;
            for (i=0; i<strlen(fixed_str); i++)
                    concat_str[k++] = fixed_str[i];

            concat_str[k] = '\0'; // string ends with '\0'

            // Close both reading ends
            close(fd1[0]);
            close(fd2[0]);

            // Write concatenated string and close writing end
            write(fd2[1], concat_str, strlen(concat_str)+1);
            close(fd2[1]);

            exit(0);
        }
}
```