**OS\_LAB\_06\_Assignment**

**CE\_054**

Aim **:-** Inter Process Communication. (Use of pipe system call and mkfifo).

Explanation :-

1. PIPE system call :-

Syntax :-

#include <unistd.h>

int pipe(int pipefd[2]);

Description :-

* Conceptually, 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 processes(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 the other process reads from the pipe. It opens a pipe, which is an area of main memory that is treated as a “virtualfile”.
* If a process tries to read before something is written to the pipe, the process is suspended until something is written.
* The array pipefd is used to return two file descriptors referring to the ends of the pipe.
* pipefd[0] refers to the read end of the pipe.
* pipefd[1] refers to the write end of the pipe.
* On success, zero is returned. On error, -1 is returned, and errno is set appropriately.

1. CLOSE system call :-

Syntax :-

#include <unistd.h>

int close(int fd);

Description :-

* close() closes a file descriptor, so that it no longer refers to any file and may be reused. Any record locks (see fcntl(2)) held on the file it was associated with, and owned by the process, are removed (regardless of the file descriptor that was used to obtain the lock).
* If fd is the last copy of a particular file descriptor the resources associated with it are freed; if the descriptor was the last reference to a file which has been removed using unlink(2) the file is deleted.
* close() returns zero on success. On error, -1 is returned, and *errno* is set appropriately.

Programs :-

1. Write a program to create a pipe and print the values of pipe file descriptors.

Code :-

// Author : Dhruv B Kakadiya

#include<stdio.h>

#include<unistd.h>

int main()

{

    int pipefd[2];

    pipe(pipefd);

    printf("\n1st pipe :  %d %d\n", pipefd[0], pipefd[1]);

    pipe(pipefd);   // 2nd pipe

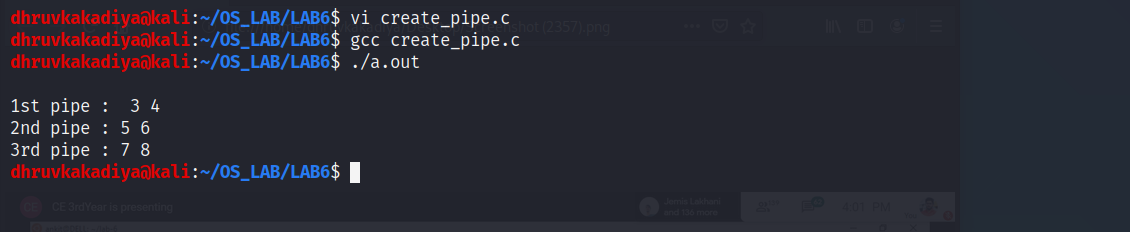
    printf("2nd pipe : %d %d\n", pipefd[0], pipefd[1]);

    pipe(pipefd);   // 3rd pipe

    printf("3rd pipe : %d %d\n", pipefd[0], pipefd[1]);

}

Output :-



1. Write a program to pass a message from parent process to child process through a

pipe.

Code :-

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<string.h>

int main()

{

    int pipefd[2], p, pid, status;

    char data[100];

    p = pipe(pipefd);

    if (p == -1)

    {

        printf("Error in creation of pipe!\n\n");

    }

    else

    {

        pid = fork();

        if (pid == -1)

        {

            printf("Error in creation of process!\n\n");

        }

        else if (pid > 0)

        {

            close(pipefd[0]);

            bzero(data, sizeof(data));

            int n = read(0, data, sizeof(data));

            data[strlen(data) - 1] = '\0';

            printf("Parent pid : %d ---> sending data : %s ---> to Child pid : %d\n\n", getpid(), data, pid);

            write(pipefd[1], data, n);

        }

        else

        {

            close(pipefd[1]);

            bzero(data, sizeof(data));

            read(pipefd[0], data, sizeof(data));

            data[strlen(data)] = '\0';

            printf("Child pid : %d ---> received data : \n\n%s ---> from Parent pid : %d\n\n", getpid(), data, getppid());

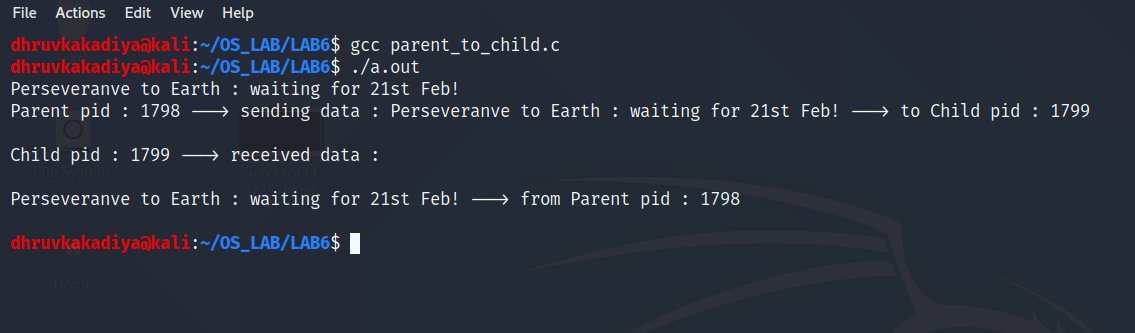
        }

    }

    wait(&status);

}

Output :-



1. Write a program to pass a message from child process to parent process through a

pipe.

Code :-

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<string.h>

int main()

{

    int pipefd[2], p, pid, status;

    char data[100];

    p = pipe(pipefd);

    if (p == -1)

    {

        printf("Error in creation of pipe!\n\n");

    }

    else

    {

        pid = fork();

        if (pid == -1)

        {

            printf("Error in creation of process!\n\n");

        }

        else if (pid == 0)

        {

            close(pipefd[0]);

            bzero(data, sizeof(data));

            int n = read(0, data, sizeof(data));

            data[strlen(data) - 1] = '\0';

            printf("Child pid : %d ---> sending data : %s --->to Parent pid : %d\n\n", getpid(), data, getppid());

            write(pipefd[1], data, n);

        }

        else

        {

            close(pipefd[1]);

            wait(NULL);

            bzero(data, sizeof(data));

            int n = read(pipefd[0], data, sizeof(data));

            data[strlen(data)] = '\0';

            printf("Parent pid : %d ---> received data : %s ---> from Child pid : %d\n\n", getpid(), data, pid);

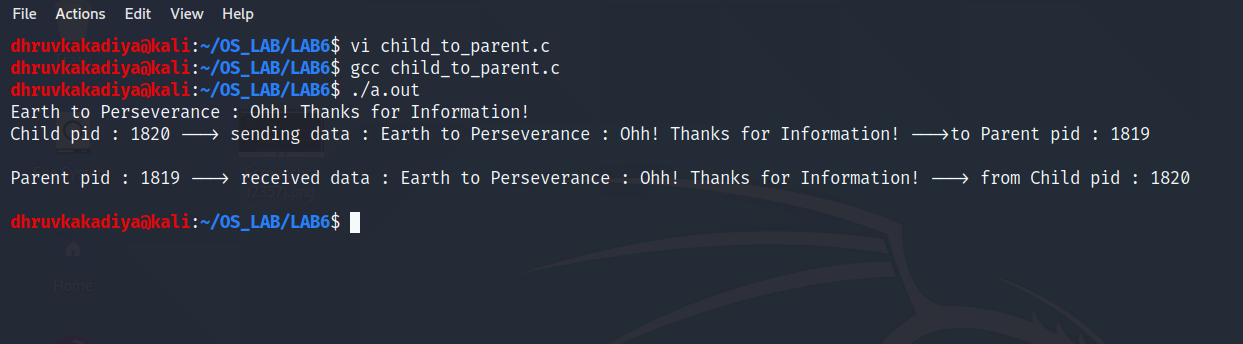
        }

    }

    wait(&status);

}

Output :-



1. Write a program to pass file name from parent process to child process through a

pipe, child process should open file and print content of that file.

Code :-

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<sys/types.h>

#include<string.h>

int main(int argc, char\* argv[])

{

    int pipefd[2], p, pid, n, status;

    char file\_name[20];

    char content[100];

    p = pipe(pipefd);

    if (p == -1)

    {

        printf("Error in creation of pipe!\n");

    }

    else

    {

        pid  = fork();

        if (pid == -1)

        {

            printf("Error in creation of process!\n");

        }

        else if (pid > 0)

        {

            close(pipefd[0]);

            bzero(file\_name, sizeof(file\_name));

            n = read(0, file\_name, sizeof(file\_name));

            file\_name[strlen(file\_name) - 1] = '\0';

            printf("parent pid : %d ---> sending file\_name : %s ---> to child pid : %d\n\n", getpid(), file\_name, pid);

            write(pipefd[1], file\_name, n);

        }

        else

        {

            close(pipefd[1]);

            bzero(file\_name, sizeof(file\_name));

            read(pipefd[0], file\_name, sizeof(file\_name));

            file\_name[strlen(file\_name)] = '\0';

            int fd = open(file\_name, O\_RDONLY);

            bzero(content, sizeof(content));

            read(fd, content, sizeof(content));

            printf("chlid pid : %d ---> received msg : \n\n%s\n ---> from parent pid : %d\n", getpid(), content, getppid());

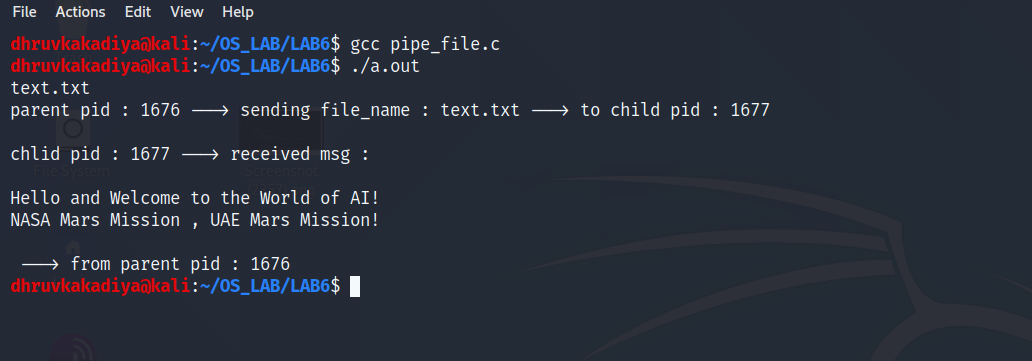
        }

    }

    wait(&status);

}

Output :-



1. Write a program to pass file name from parent process to child process through a

pipe, child process should pass the file contents to parent process and parent should

print the contents.

Code :-

#include<stdio.h>

#include<unistd.h>

#include<sys/stat.h>

#include<sys/wait.h>

#include<string.h>

#include<fcntl.h>

int main()

{

    int pipefd[2], pipefd2[2],p, pid, p2, status;

    char file\_name[20], content[100], last\_content[100];

    p = pipe(pipefd);

    p2 = pipe(pipefd2);

    if (p == -1)

    {

        printf("Error in creation in pipe!\n\n");

    }

    else

    {

        pid = fork();

        if (pid == -1)

        {

            printf("Errro in creation of process!\n\n");

        }

        else if (pid > 0)

        {

            close(pipefd[0]);

            bzero(file\_name, sizeof(file\_name));

            int n = read(0, file\_name, sizeof(file\_name));

            file\_name[strlen(file\_name) - 1] = '\0';

            printf("Parent pid : %d ---> sending file\_name : %s ---> to Child pid : %d\n\n\n", getpid(), file\_name, pid);

            write(pipefd[1], file\_name, n);

            wait(&status);

            close(pipefd2[1]);

            bzero(last\_content, sizeof(last\_content));

            read(pipefd2[0], last\_content, sizeof(last\_content));

            file\_name[strlen(last\_content)] = '\0';

            printf("parent pid : %d ---> received data : \n%s ---> from Child pid : %d\n\n\n", getpid(), last\_content, pid);

        }

        else

        {

            close(pipefd[1]);

            bzero(file\_name, sizeof(file\_name));

            read(pipefd[0], file\_name, sizeof(file\_name));

            last\_content[strlen(file\_name)] = '\0';

            int fd = open(file\_name, O\_RDONLY);

            bzero(content, sizeof(content));

            int n = read(fd, content, sizeof(content));

            printf("Child pid : %d ---> received data : \n%s\n ---> from Parent pid : %d\n\n\n", getpid(), content, getppid());

            if (p2 == -1)

            {

                printf("Error in creation of Process!\n");

            }

            else

            {

                close(pipefd2[0]);

                printf("Child pid : %d ---> sending data : \n%s\n ---> to parent pid : %d\n\n", getpid(), content, getppid());

                write(pipefd2[1], content, n);

            }

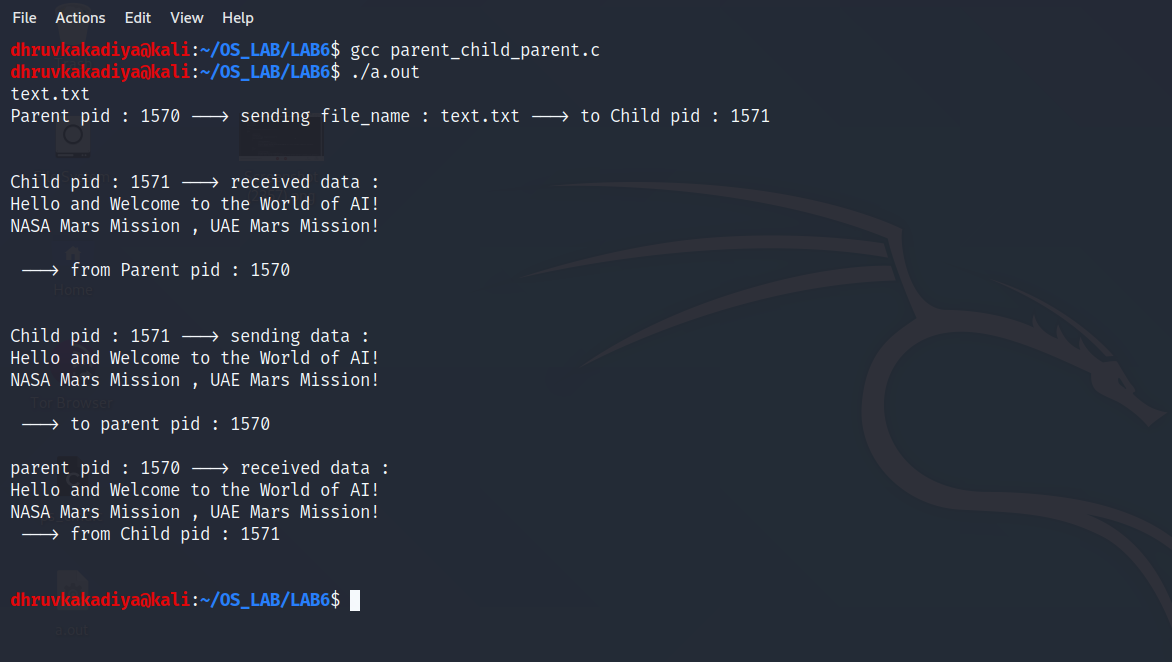
        }

    }

    return 0;

}

Output :-



1. Write a program to implement ‘mkfifo’ command.

Code :-

* firstfifo.c file

#include<stdio.h>

#include<string.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

    mkfifo("filefifo", 0777);

    char str2[100];

    while(1)

    {

        int fd = open("filefifo", O\_WRONLY);

        fgets(str2, 100, stdin);

        write(fd, str2, strlen(str2)+1);

        close(fd);

    }

    return 0;

}

* secondfifo.c file

#include<stdio.h>

#include<string.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

    int fd2;

    mkfifo("filefifo", 0777);

    char array1[100];

    while(1)

    {

        fd2 = open("filefifo", O\_RDONLY);

        read(fd2, array1, 100);

        printf("%s\n", array1);

        close(fd2);

    }

    return 0;

}

Output :-

