**Subject : Operating Systems**

**Lab 7 Assignment**

**Date : 30/08/2020**

**Name : Devang Bhalala**

**Roll no: 007**

Aim : Study of Dup2 system call and Execl function.

**Theory Explanation:**

1. **dup2 system call:**

**SYNOPSIS**

#include <unistd.h>

int dup2(int oldfd, int newfd);

**DESCRIPTION**

This system call creates a copy of the file descriptor oldfd.

dup2() makes newfd be the copy of oldfd, closing newfd first if necessary, but note the following:

\* If oldfd is not a valid file descriptor, then the call fails, and newfd is not closed.

\* If oldfd is a valid file descriptor, and newfd has the same value as oldfd, then dup2() does nothing, and returns newfd.

After a successful return from this system call, the old and new file descriptors may be used interchangeably. They refer to the same open file description and thus share file offset and file status flags.

The two descriptors do not share file descriptor flags (the close-on-exec flag). The close-on-exec flag for the duplicate descriptor is off.

**RETURN VALUE**

On success, this system call returns the new descriptor. On error, -1 is returned, and errno is set appropriately.

1. **execl function:**

**SYNOPSIS**

#include <[unistd.h](https://linux.die.net/include/unistd.h)>

extern char \*\*environ;

int execl(const char \*path, const char \*arg, ...);

**DESCRIPTION**

The const char \*arg and subsequent ellipses in the execl(), execlp(), and execle() functions can be thought of as arg0, arg1, ..., argn. Together they describe a list of one or more pointers to null-terminated strings that represent the argument list available to the executed program.

The first argument, by convention, should point to the filename associated with the file being executed.

The list of arguments must be terminated by a NULL pointer, and, since these are variadic functions, this pointer must be cast (char \*) NULL.

**RETURN VALUE**

The execl() functions only return if an error has occurred. The return value is -1, and errno is set to indicate the error.

**ASSIGNMENT PROGRAMS:**

**PROGRAM 1:**

Write a program to implement ls | sort functionality using the system calls and functions covered in the lab.

**CODE:**

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<string.h>

void main()

{

    int pid,pret,pipefd[2],wst;

    pret = pipe(pipefd);

    if (pret == -1)

    {

        printf("\n\n--Task failed. Could not create pipe.--\n\n");

    }

    else

    {

        pid = fork();

        if (pid == -1)

        {

            printf("\n\n--Task failed. Could not create pipe.--\n\n");

        }

        else if (pid > 0)

        {

            wait(&wst);

            close(pipefd[1]);

            printf("---Entered in Parent process---\n\n");

            dup2(pipefd[0], 0);

            execl("/bin/sort","sort",(char\*)NULL);

        }

        else{

            close(pipefd[0]);

            printf("---Entered in Child process---\n");

            dup2(pipefd[1], 1 );

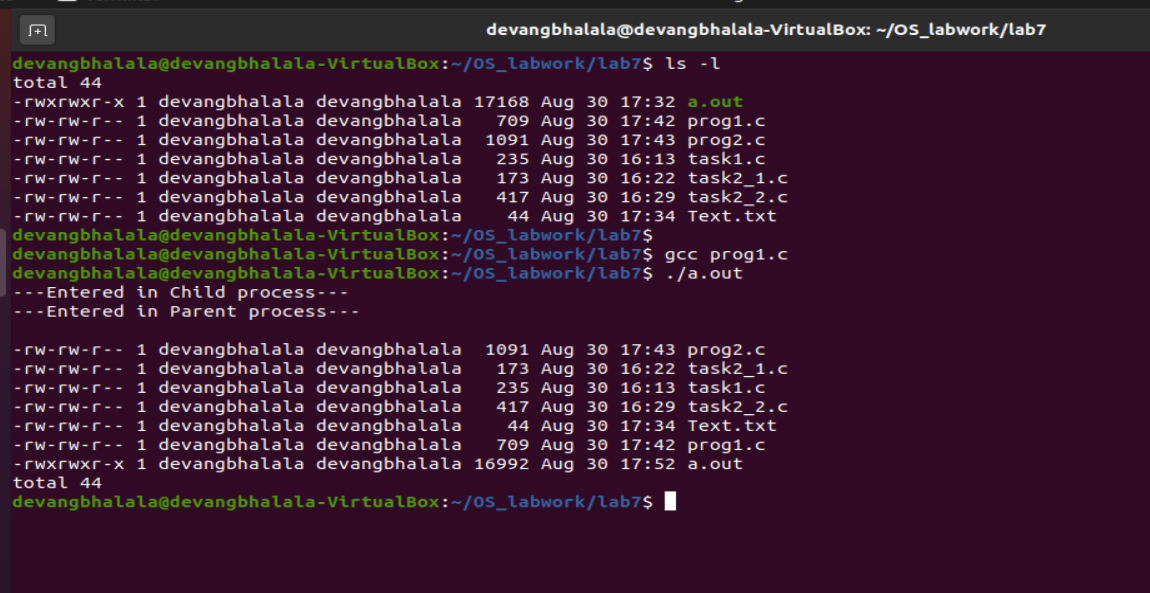
            execl("/bin/ls","ls","-l",(char\*)NULL);

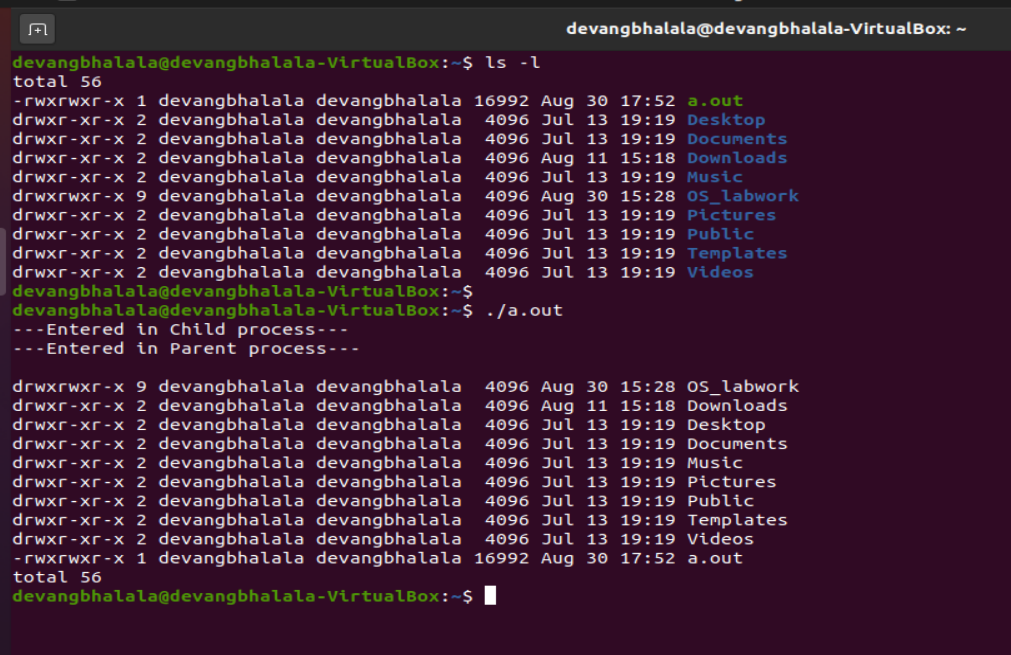
        }

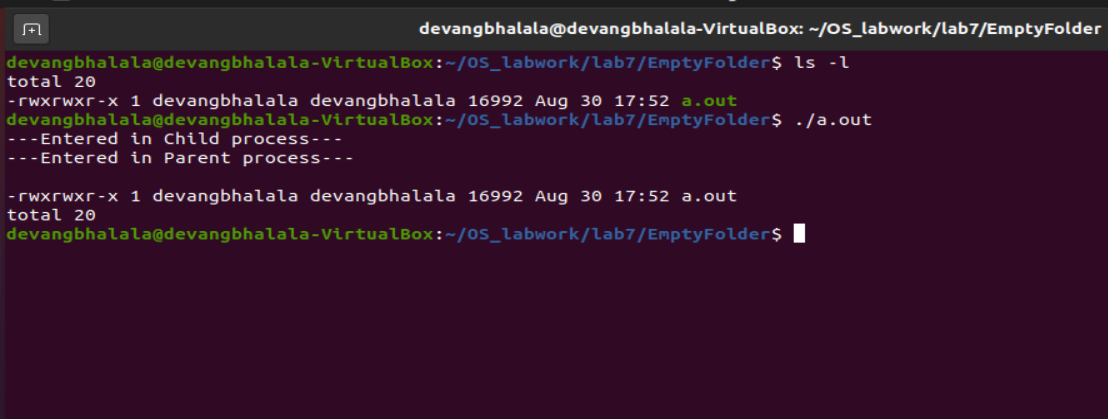
    }

}

**OUTPUT:**







**PROGRAM 2:**

Write a program to achieve following:

=> Child process should open a file with the contents to be sorted, pass the contents to parent process.

=> Parent process should sort the contents of the file and display.

**CODE:**

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<string.h>

void main()

{

    int pid,pret,pipefd[2],wst,filehand,rd;

    char filedata[1000];

    char filenm[100];

    pret = pipe(pipefd);

    if (pret == -1)

    {

        printf("\n\n--Task failed. Could not create pipe.--\n\n");

    }

    else

    {

        pid = fork();

        if (pid == -1)

        {

            printf("\n\n--Task failed. Could not create pipe.--\n\n");

        }

        else if (pid > 0)

        {

            wait(&wst);

            close(pipefd[1]);

            printf("\n---Entered in Parent process---\n\n");

            dup2(pipefd[0], 0);

            execl("/bin/sort","sort",(char\*)NULL);

        }

        else{

            close(pipefd[0]);

            printf("---Entered in Child process---\n");

            printf("Enter File name to sort : \n");

            bzero(filenm, sizeof(filenm));

            rd = read(0, filenm, sizeof(filenm));

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

            filehand = open(filenm, O\_RDONLY);

            bzero(filedata, sizeof(filedata));

            rd = read(filehand, filedata, sizeof(filedata));

            write(pipefd[1], filedata, rd);

        }

    }

}

**OUTPUT:**



