| **Fr. Conceicao Rodrigues College of Engineering**  **Department of Computer Engineering** | | | |
| --- | --- | --- | --- |
| **Student’s Roll No** | **9913** | **Students Name** | **Mark lopes** |
| **Date of Performance** |  | **SE Computer – Div** | **A** |

**Aim:** To study Process and File Management System Calls

**Lab Outcome:**

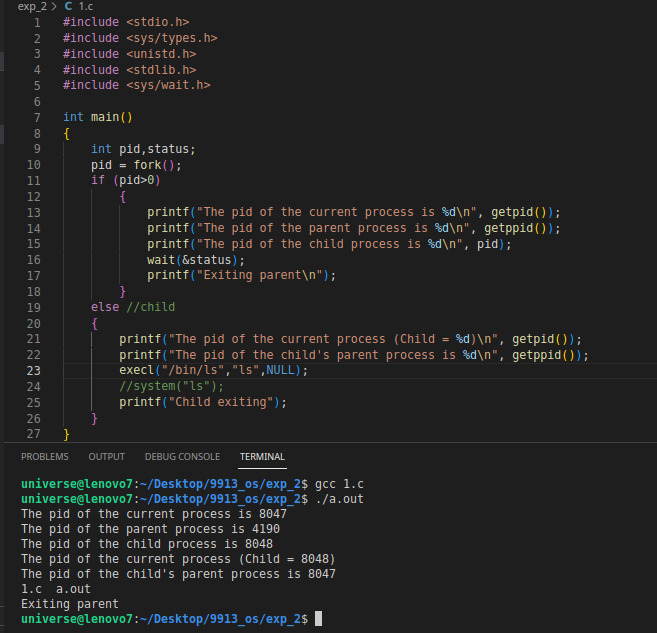
**CSL403.1: Demonstrate basic Operating system Commands, Shell scripts, System Calls and API wrt Linux.**

**Problem Statements:**

(1.) Process related System Calls.

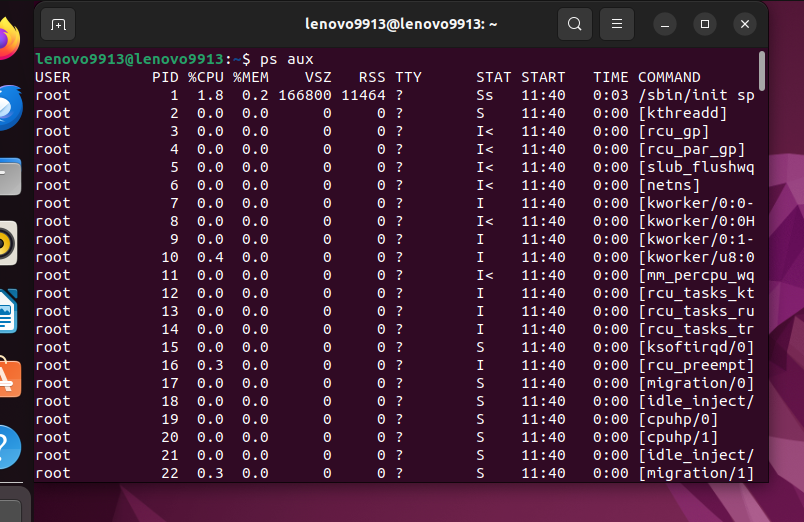
a) Create a child process in Linux using the fork system call. From the child process obtain the process ID of both child and parent by using getpid and getppid system call.

b) Explore wait and waitpid before termination of process.



c) Explain ps command and output in detail. What is Zombie and Orphan Process? Show the output.

The ps command in Unix-like operating systems is used to display information about active processes. It provides a snapshot of the current processes running on the system.



Zombie Process:

1. A Zombie process is a process that has completed execution but still has an entry in the process table.
2. The process entry is retained to allow the parent process to retrieve information about the child after its termination.

Orphan Process:

1. An Orphan process is a child process that is still running after its parent process has terminated.
2. Orphan processes are adopted by the init process (usually with PID 1), which becomes their new parent.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main() {

pid\_t child\_pid = fork();

if (child\_pid > 0) {

// Parent process

printf("Parent process (PID=%d)\n", getpid());

sleep(10); // Sleep to give the child process time to become a Zombie

} else if (child\_pid == 0) {

// Child process

printf("Child process (PID=%d)\n", getpid());

exit(0); // Child process exits immediately

} else {

// Fork failed

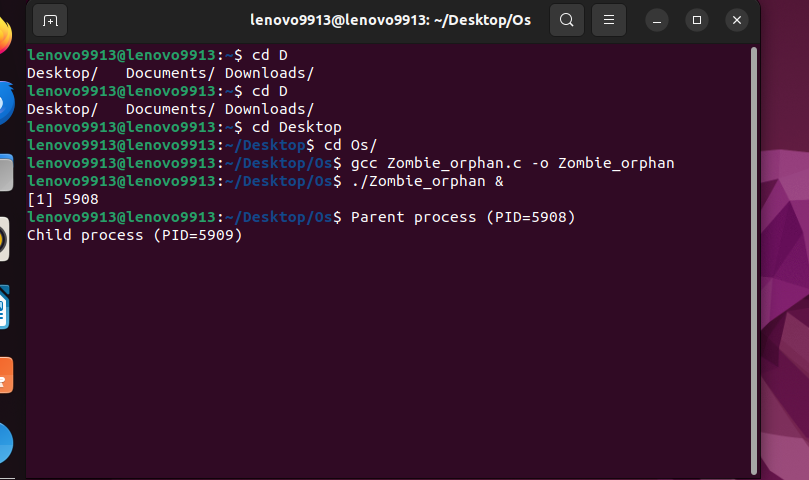
perror("Fork failed");

exit(1);

}

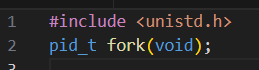
return 0;

}

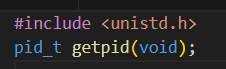


d) Explain fork(), getpid(), getppid(),wait() and waitpid() with syntax.

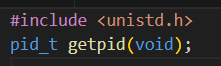
The fork() function is used to create a new process by duplicating the existing process. The new process is called the child process, and the existing process is called the parent process.



The getpid() function returns the process ID (PID) of the calling process.



The getppid() function returns the parent process ID (PID) of the calling process.



The wait() function is used by a parent process to wait for the termination of its child process. It suspends the execution of the calling process until one of its child processes exits.



The waitpid() function is a more flexible version of wait() that allows you to wait for a specific child process and provides additional options.



(2) File related system calls

a) Program to copy contents of one file (source) to another file (destination). Finally displaying contents of destination file.

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <fcntl.h>

#include <string.h>

int main()

{

int choice;

char name[50], name1[50];

char buff[500];

char buff1[500];

up:

printf("Enter the choice: \n1.Create\n2.Write and Read\n3.Copy\n4.Exit");

scanf("%d", &choice);

switch (choice)

{

case 1: // create file

printf("Enter filename: ");

scanf("%s", name);

int fd = open(name, O\_CREAT | O\_EXCL | O\_WRONLY, 0666);

if (fd == -1)

{

printf("Error");

exit(1);

}

else

{

printf("File created successfully");

close(fd);

goto up;

}

case 2: // write and read

printf("Enter the name of the file: ");

scanf("%s", name);

fd = open(name, O\_RDWR);

if (fd == -1)

{

printf("Error");

}

printf("Enter the data to be stored: ");

scanf("%s", buff);

write(fd, buff, strlen(buff));

lseek(fd, 0, SEEK\_SET);

read(fd, buff1, sizeof(buff1));

printf("contents are %s\n", buff1);

close(fd);

goto up;

case 3:

printf("Enter the name of the file you want to copy from: ");

scanf("%s", name1);

printf("Enter the name of the file you want to copy to: ");

scanf("%s", name);

fd = open(name1, O\_RDWR);

if (fd == -1)

{

printf("Error");

}

read(fd, buff1, sizeof(buff1));

close(fd);

fd = open(name, O\_RDWR | O\_CREAT | O\_TRUNC, 0666);

if (fd == -1)

{

printf("Error");

}

write(fd, buff1, strlen(buff1));

close(fd);

break;

case 4: // exit

printf("Exiting program\n");

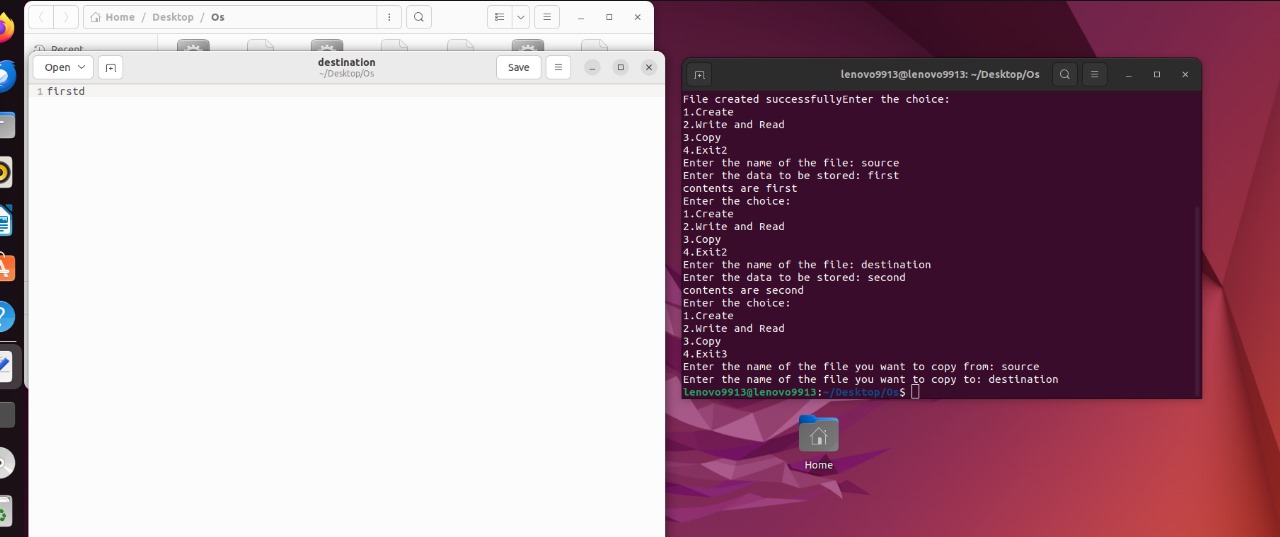
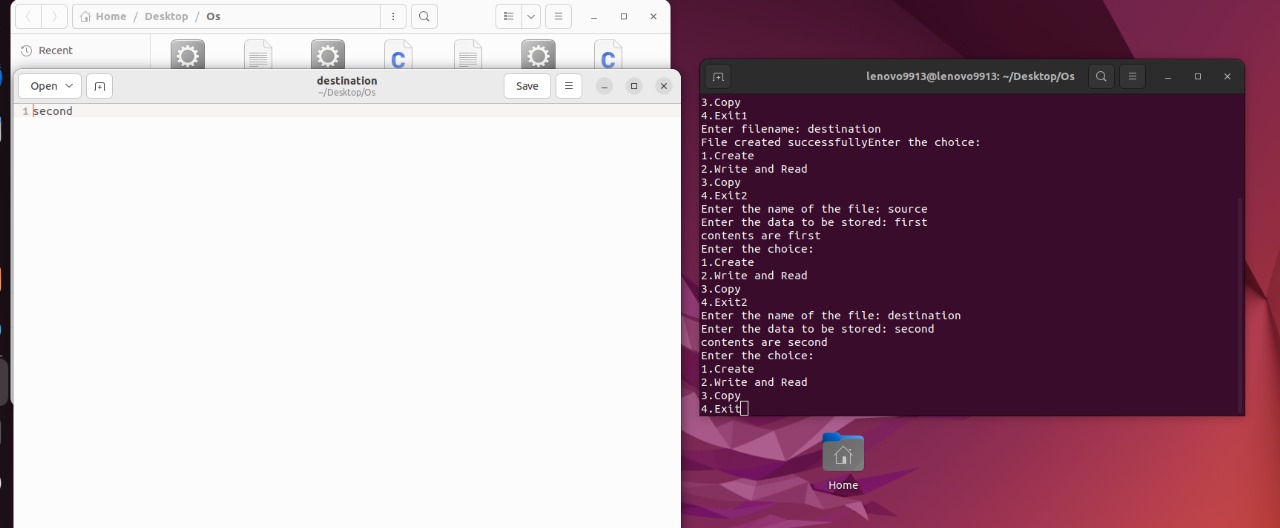
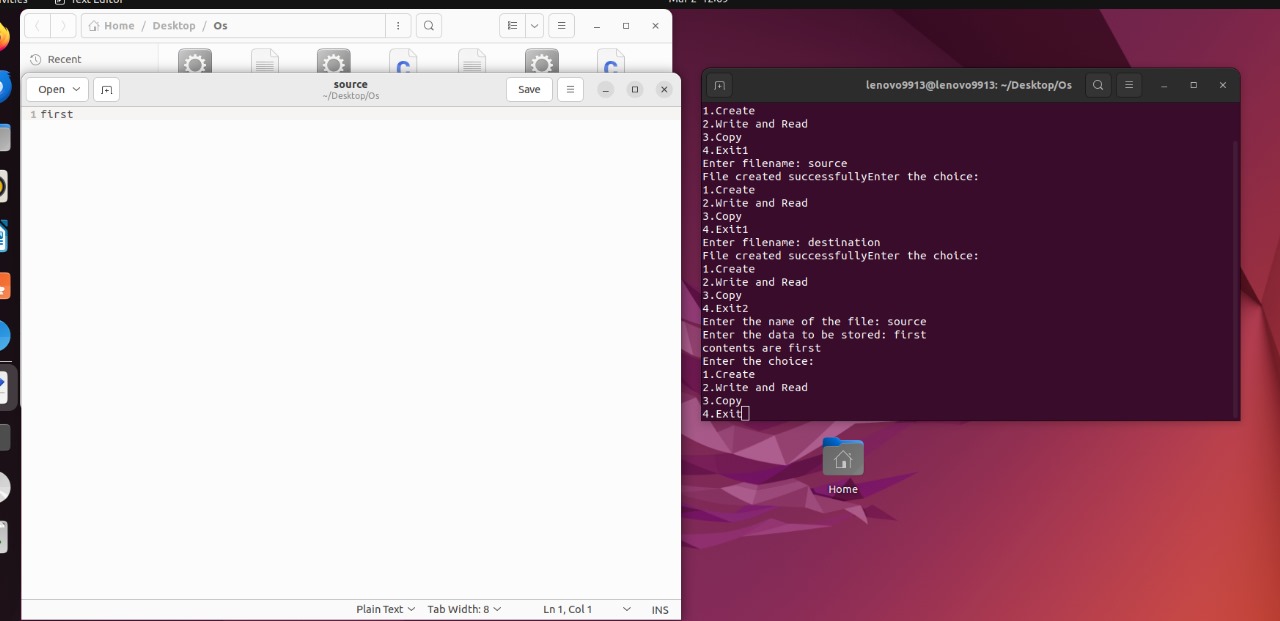
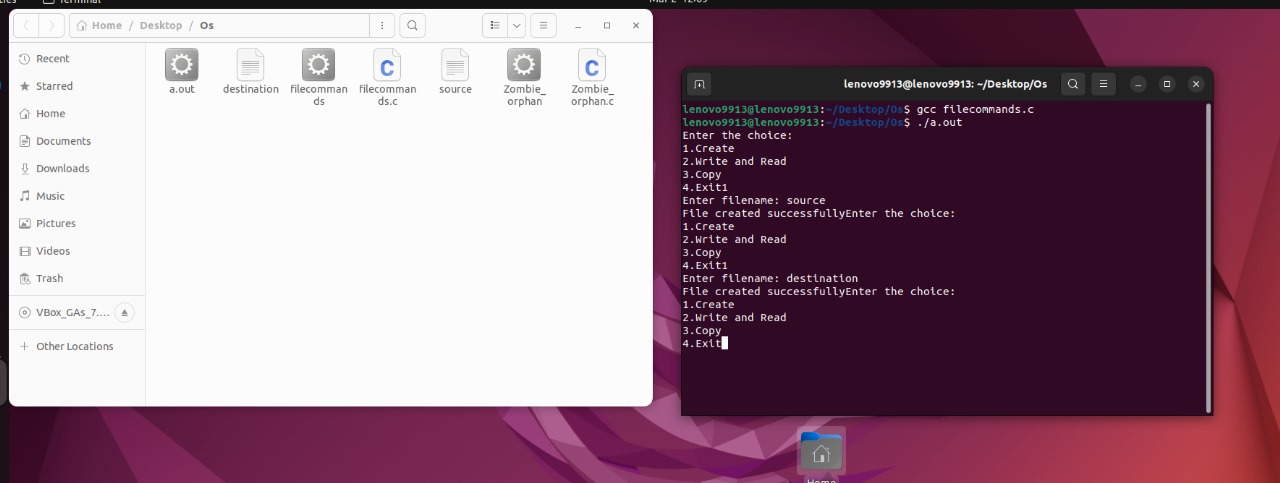
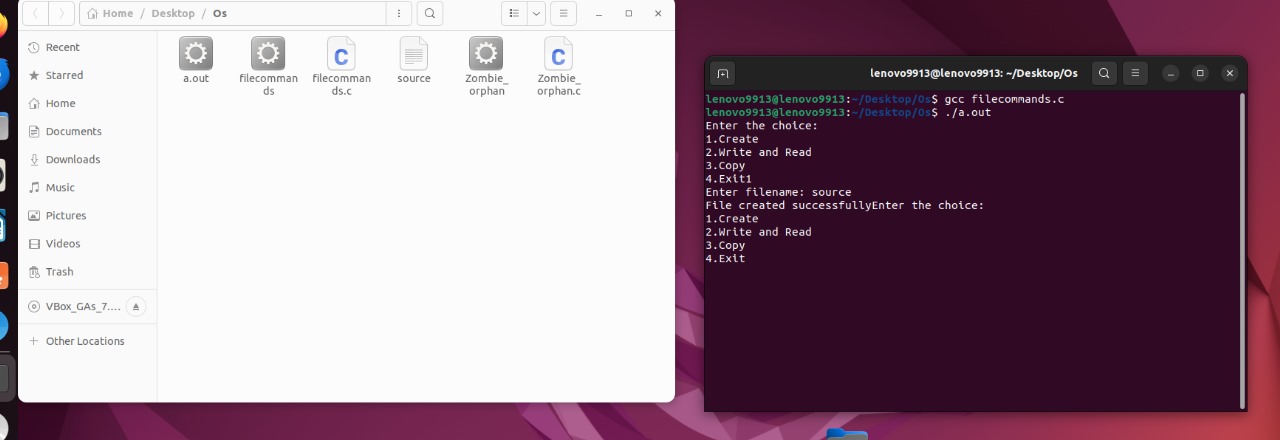
exit(EXIT\_SUCCESS);

break;

}

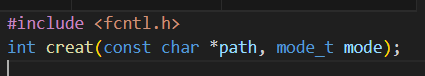
return 0;

}

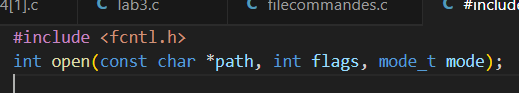


b) 2. Explain creat(), open(), close(), read() and write() with syntax.

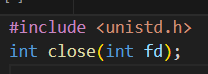
The creat() function is used to create a new file or open an existing file for writing. If the file already exists, it will be truncated to zero length.



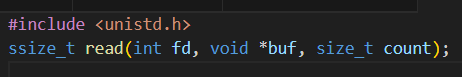
The open() function is used to open an existing file or create a new file and return a file descriptor.



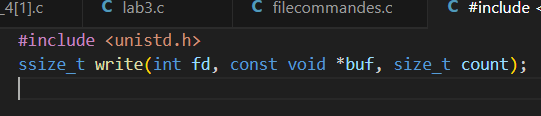
The close() function is used to close a file descriptor.



The read() function is used to read data from an open file descriptor into a buffer.



The write() function is used to write data to an open file descriptor from a buffer.



**References:**

<https://www.geeksforgeeks.org/fork-system-call/>

<https://www.geeksforgeeks.org/getppid-getpid-linux/>

<https://www.geeksforgeeks.org/wait-system-call-c/>

<https://www.geeksforgeeks.org/zombie-and-orphan-processes-in-c/>

<https://www.geeksforgeeks.org/input-output-system-calls-c-create-open-close-read-write/>

| **On time Submission(2)** | **Knowledge of Topic(4)** | **Implementation and Demonstraion(4)** | **Total (10)** |
| --- | --- | --- | --- |
|  |  |  |  |
| **Signature of Faculty** |  | **Date of Submission** |  |