

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.

- 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.
- Explore wait and waitpid before termination of process.

```

exp_2 > C 1.c
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5  #include <sys/wait.h>
6
7  int main()
8  {
9      int pid,status;
10     pid = fork();
11     if (pid>0)
12     {
13         printf("The pid of the current process is %d\n", getpid());
14         printf("The pid of the parent process is %d\n", getppid());
15         printf("The pid of the child process is %d\n", pid);
16         wait(&status);
17         printf("Exiting parent\n");
18     }
19     else //child
20     {
21         printf("The pid of the current process (Child = %d)\n", getpid());
22         printf("The pid of the child's parent process is %d\n", getppid());
23         execl("/bin/ls","ls",NULL);
24         //system("ls");
25         printf("Child exiting");
26     }
27 }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

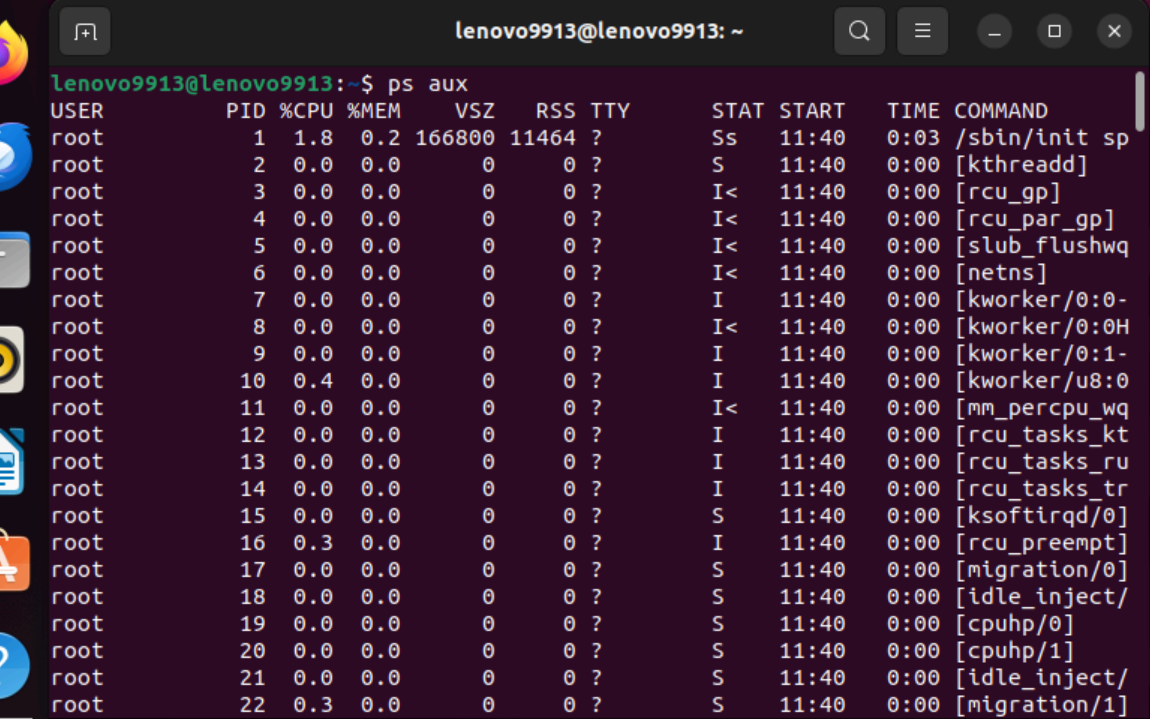
```

universe@lenovo7:~/Desktop/9913_os/exp_2$ gcc 1.c
universe@lenovo7:~/Desktop/9913_os/exp_2$ ./a.out
The pid of the current process is 8047
The pid of the parent process is 4190
The pid of the child process is 8048
The pid of the current process (Child = 8048)
The pid of the child's parent process is 8047
l.c a.out
Exiting parent
universe@lenovo7:~/Desktop/9913_os/exp_2$

```

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.



```
lenovo9913@lenovo9913: ~$ ps aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	1.8	0.2	166800	11464	?	Ss	11:40	0:03	/sbin/init sp
root	2	0.0	0.0	0	0	?	S	11:40	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	I<	11:40	0:00	[rcu_gp]
root	4	0.0	0.0	0	0	?	I<	11:40	0:00	[rcu_par_gp]
root	5	0.0	0.0	0	0	?	I<	11:40	0:00	[slub_flushwq]
root	6	0.0	0.0	0	0	?	I<	11:40	0:00	[netns]
root	7	0.0	0.0	0	0	?	I	11:40	0:00	[kworker/0:0-
root	8	0.0	0.0	0	0	?	I<	11:40	0:00	[kworker/0:0H
root	9	0.0	0.0	0	0	?	I	11:40	0:00	[kworker/0:1-
root	10	0.4	0.0	0	0	?	I	11:40	0:00	[kworker/u8:0
root	11	0.0	0.0	0	0	?	I<	11:40	0:00	[mm_percpu_wq
root	12	0.0	0.0	0	0	?	I	11:40	0:00	[rcu_tasks_kt
root	13	0.0	0.0	0	0	?	I	11:40	0:00	[rcu_tasks_ru
root	14	0.0	0.0	0	0	?	I	11:40	0:00	[rcu_tasks_tr
root	15	0.0	0.0	0	0	?	S	11:40	0:00	[ksoftirqd/0]
root	16	0.3	0.0	0	0	?	I	11:40	0:00	[rcu_preempt]
root	17	0.0	0.0	0	0	?	S	11:40	0:00	[migration/0]
root	18	0.0	0.0	0	0	?	S	11:40	0:00	[idle_inject/
root	19	0.0	0.0	0	0	?	S	11:40	0:00	[cpuhp/0]
root	20	0.0	0.0	0	0	?	S	11:40	0:00	[cpuhp/1]
root	21	0.0	0.0	0	0	?	S	11:40	0:00	[idle_inject/
root	22	0.3	0.0	0	0	?	S	11:40	0:00	[migration/1]

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.

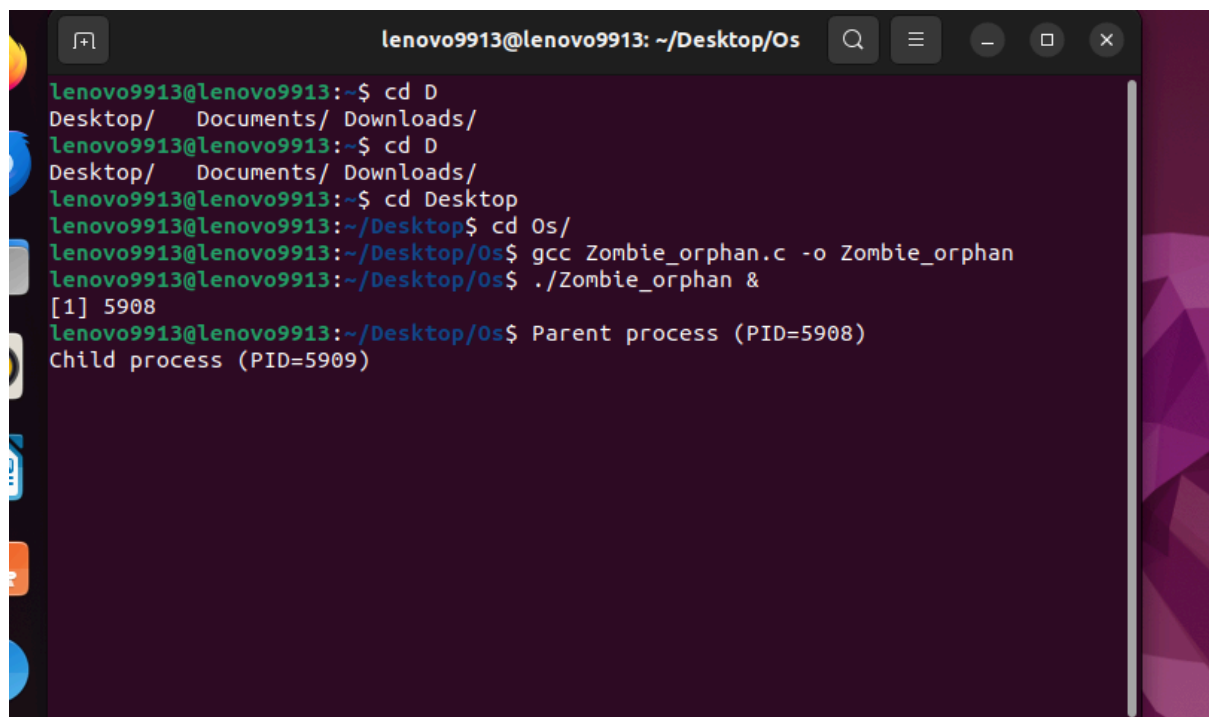
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

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;
}
```

A terminal window titled 'lenovo9913@lenovo9913: ~/Desktop/Os' showing the execution of a C program. The user navigates to the Desktop directory, then to a subdirectory 'Os'. They compile the file 'Zombie\_orphan.c' using 'gcc' with the output file 'Zombie\_orphan'. Then, they run the program with './Zombie\_orphan &'. The output shows the parent process (PID=5908) printing its PID and the child process (PID=5909) printing its PID. The terminal has a dark purple background and a light blue cursor.

```
lenovo9913@lenovo9913:~$ cd D
Desktop/  Documents/ Downloads/
lenovo9913@lenovo9913:~$ cd D
Desktop/  Documents/ Downloads/
lenovo9913@lenovo9913:~$ cd Desktop
lenovo9913@lenovo9913:~/Desktop$ cd Os/
lenovo9913@lenovo9913:~/Desktop/Os$ gcc Zombie_orphan.c -o Zombie_orphan
lenovo9913@lenovo9913:~/Desktop/Os$ ./Zombie_orphan &
[1] 5908
lenovo9913@lenovo9913:~/Desktop/Os$ Parent process (PID=5908)
Child process (PID=5909)
```

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.

```
1  #include <unistd.h>
2  pid_t fork(void);
```

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

```
#include <unistd.h>
pid_t getpid(void);
```

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

```
#include <unistd.h>
pid_t getppid(void);
```

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.

```
#include <sys/wait.h>
pid_t wait(int *status);
```

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

```
#include <sys/wait.h>
pid_t waitpid(pid_t pid, int *status, int 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;

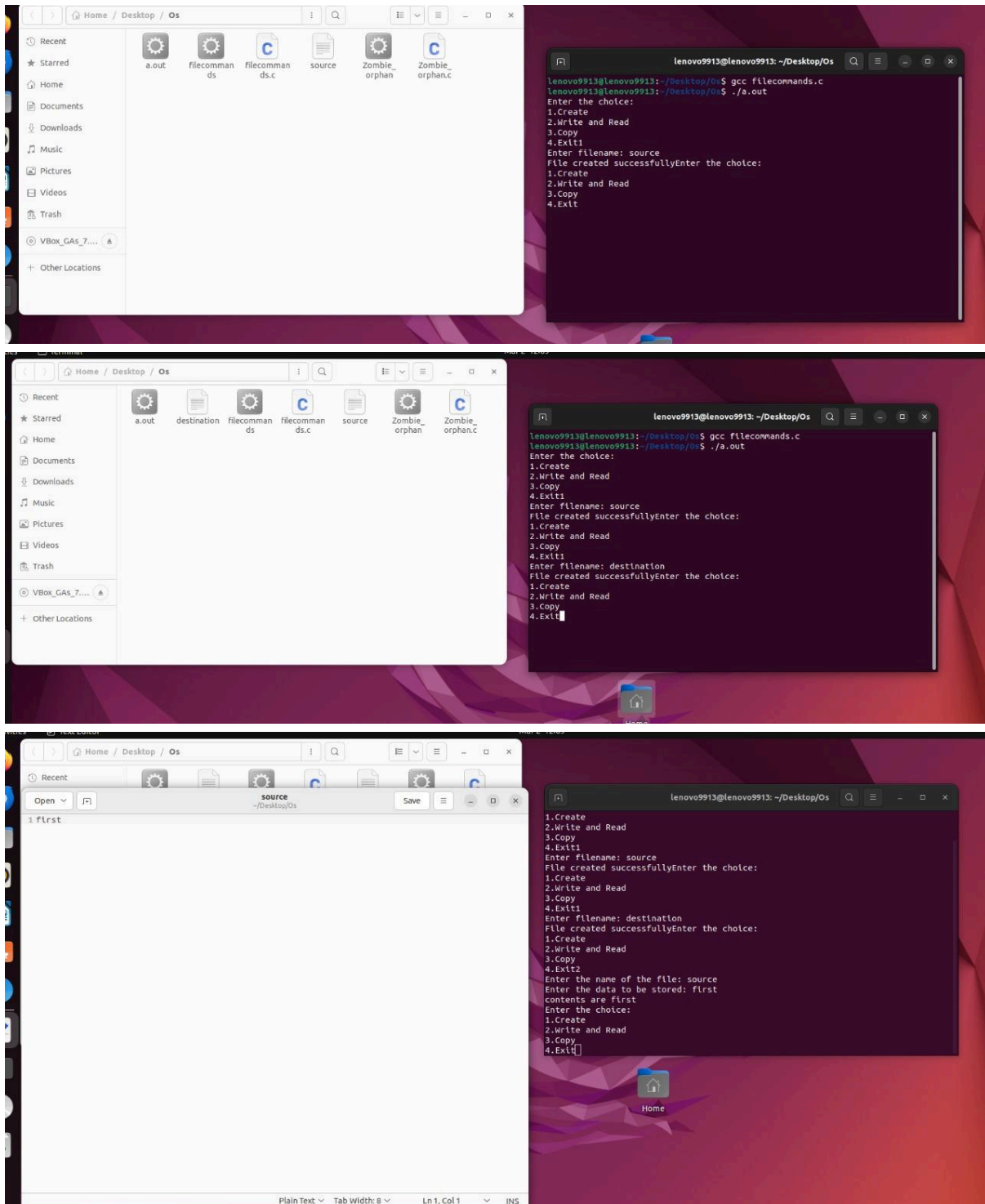
    case 3:
        printf("Enter the name of the file you want to copy from: ");
        scanf("%s", name1);
        printf("Enter the name of the file you want to copy to: ");
        scanf("%s", name);

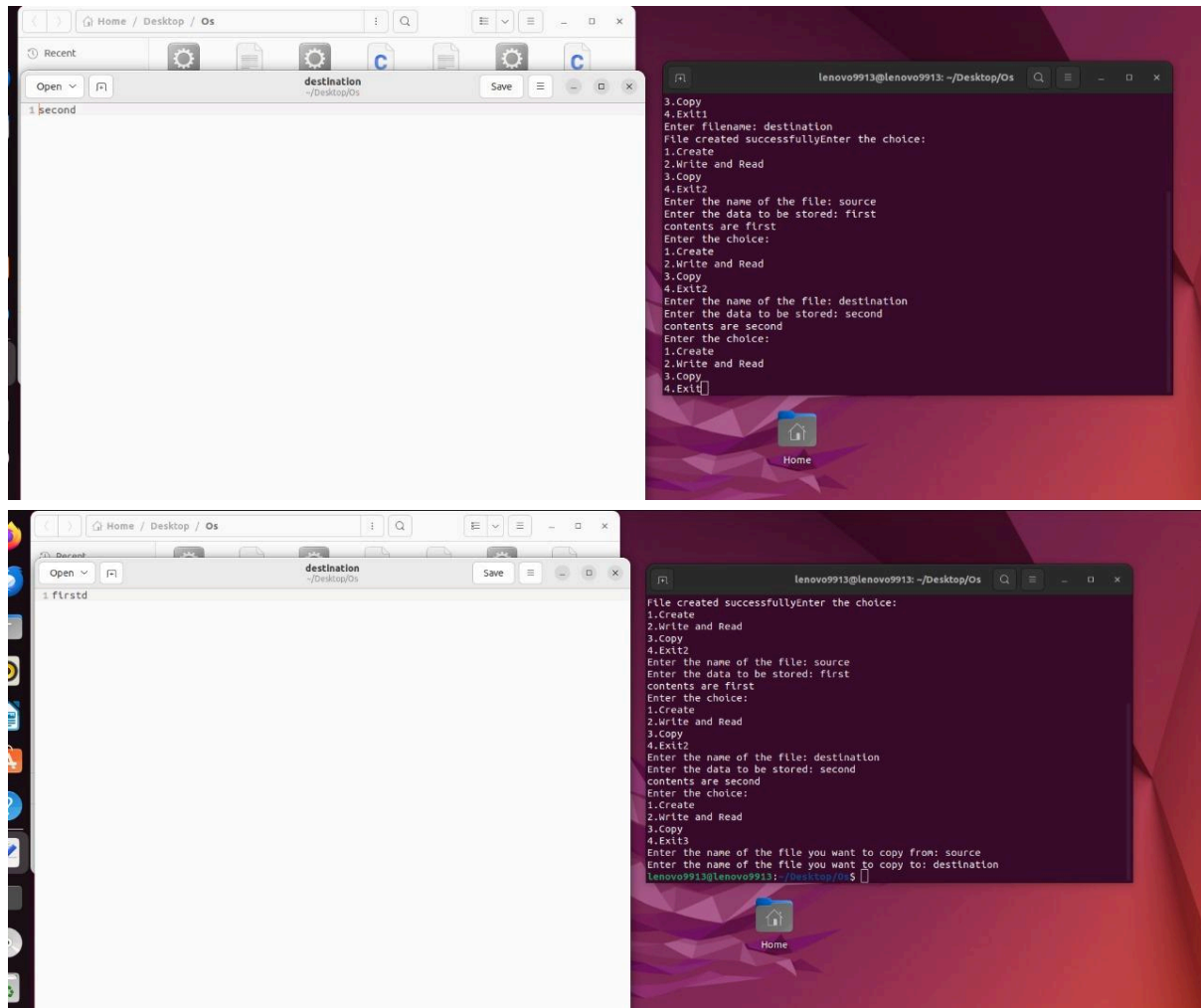
        fd = open(name1, O_RDWR);
        if (fd == -1)
        {
            printf("Error");
        }
        read(fd, buff1, sizeof(buff1));
        close(fd);

        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.

```
#include <fcntl.h>
int creat(const char *path, mode_t mode);
```

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

```
#include <fcntl.h>
int open(const char *path, int flags, mode_t mode);
```

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

```
#include <unistd.h>
int close(int fd);
```



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

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

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

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

### 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	