Operating System

Submitted By: Fatima Zahra

Sap Id: 55551

Lab: No.8

Submitted To: Ma'am Ayesha Akram

Riphah International University of Islamabad

Task 1:

```
student@student-virtual-machine:~$ pico f1.c
student@student-virtual-machine:~$ gcc f1.c
student@student-virtual-machine:~$ ./f1.c
bash: ./f1.c: Permission denied
student@student-virtual-machine:~$ ./a.out
Process ID: 3679 | Loop value: 1
Process ID: 3680 | Loop value: 1
Process ID: 3681 | Loop value: 1
Process ID: 3682 | Loop value: 1
Process ID: 3680 | Loop value: 2
Process ID: 3679 | Loop value: 2
Process ID: 3681 | Loop value: 2
Process ID: 3682 | Loop value: 2
Process ID: 3679 | Loop value: 3
Process ID: 3680 | Loop value: 3
Process ID: 3682 | Loop value: 3
Process ID: 3681 | Loop value: 3
Process ID: 3679 | Loop value: 4
Process ID: 3680 | Loop value: 4
Process ID: 3681 | Loop value: 4
Process ID: 3682 | Loop value: 4
Process ID: 3679 | Loop value: 5
Process ID: 3680 | Loop value: 5
Process ID: 3682 | Loop value: 5
```

```
  GNU nano 6.2                                f1.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>

int main(void) {
    pid_t pid1, pid2;

    pid1 = fork();
    if (pid1 < 0) {
        perror("fork failed");
        return 1;
    }

    pid2 = fork();
    if (pid2 < 0) {
        perror("fork failed");
        return 1;
    }
```

Task 2:

```
student@student-virtual-machine:~$ pico f2.c
student@student-virtual-machine:~$ gcc f2.c
student@student-virtual-machine:~$ ./a.out
Child 1 → PID: 3752 | Parent PID: 3751
Child 2 → PID: 3753 | Parent PID: 3751
Child 3 → PID: 3754 | Parent PID: 3751
Parent Process → PID: 3751
student@student-virtual-machine:~$
```

```
  GNU nano 6.2                                f2.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>

int main(void) {
    pid_t pid1, pid2, pid3;

    // First child
    pid1 = fork();
    if (pid1 < 0) {
        perror("Fork failed for child 1");
        exit(1);
    } else if (pid1 == 0) {
        printf("Child 1 → PID: %d | Parent PID: %d\n", (int)getpid(), (int)getp>
        exit(0);
    }
```

```
    pid2 = fork();
    if (pid2 < 0) {
        perror("Fork failed for child 2");
        exit(1);
    } else if (pid2 == 0) {
        printf("Child 2 → PID: %d | Parent PID: %d\n", (int)getpid(), (int)getp>
        exit(0);
    }
```

```
    pid3 = fork();
    if (pid3 < 0) {
        perror("Fork failed for child 3");
        exit(1);
    } else if (pid3 == 0) {
        printf("Child 3 → PID: %d | Parent PID: %d\n", (int)getpid(), (int)getp>
        exit(0);
    }

    // Parent waits for all children to finish
    wait(NULL);
    wait(NULL);
    wait(NULL);

    printf("Parent Process → PID: %d\n", (int)getpid());
    return 0;
}
```

Task 3:

A System Call is a way for a program to request services from the operating system's kernel, like reading files, creating processes, or managing memory.

Types of System Calls:

1. **Process Control**

Manage processes (create, execute, terminate).
fork(), exec(), wait(), exit()

2. **File Management**

Create, read, write, delete files.
open(), read(), write(), close()

3. **Device Management**

Request or release device access.
ioctl(), read(), write()

4. **Information Maintenance**

Get or set system data.
getpid(), alarm(), sleep()

5. **Communication**

Exchange information between processes.
pipe(), shmget(), msgsnd()

◆ **Examples:**

fork() → creates a new process.
exec() → replaces the current process with a new program.
wait() → parent waits for a child process to finish.
exit() → terminates the current process.