Operating Systems Laboratory

B.Tech. 5th Semester

Name : Subhendu Maji

Roll Number : 18ETCS002121

Department : Computer Science and Engineering

Faculty of Engineering & Technology

Ramaiah University of Applied Sciences

Name: Subhendu Maji Roll number: 18ETCS002121

Ramaiah University of Applied Sciences

Private University Established in Karnataka State by Act No. 15 of 2013

Faculty	Engineering & Technology
Programme	B. Tech. in Computer Science and Engineering
Year/Semester	2nd Year / 5th Semester
Name of the Laboratory	Operating Systems Laboratory
Laboratory Code	CSC306A

List of Experiments

1. Introduction to operating system- compiling and booting of Linux kernel
2. Programs using process management system calls
3. Programs using file management system calls
4. Programs based on multithreaded programming
5. Programs for process scheduling algorithms
6. Solution to producer consumer problem using Mutex and Semaphore
7. Solutions to Dining philosopher problem using Semaphore
8. Programs for deadlock avoidance algorithm
9. Programs for memory management algorithms

Name: Subhendu Maji Roll number: 18ETCS002121

Index Sheet

No.	Lab Experiment	Performing the experiment (7)	Document (7)	Viva (6)	Total Marks (20)
1	Programs using process management system calls				
2	Programs using file management system calls				
3	Programs based on multithreaded programming				
4	Programs for process scheduling algorithms				
5	Solution to Producer Consumer Problem using Semaphore and Mutex				
6	Solution to Dining Philosopher problem using Semaphore				
7	Programs for deadlock avoidance algorithm				
8	Programs for memory management algorithms				
9	Lab Internal Test conducted along the lines of SEE and valued for 50 Marks and reduced for 20 Marks				
	Total Marks				

Component 1 = Lab Internal Marks =

Signature of the Staff In-charge

Name: Subhendu Maji Roll number: 18ETCS002121

Laboratory 1

Title of the Laboratory Exercise: Programs using process management system calls

1. Introduction and Purpose of Experiment

   A system call is a programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. There are different types of system calls developed for various purposes. They are mainly classified as process management, file management, directory management. By solving the problems students will be able to apply process management system calls

   Aim and Objectives

   Aim

   - To develop programs involving process management system calls

   Objectives

   At the end of this lab, the student will be able to

   - Use different process management system calls
   - Apply different system calls wherever required
   - Create C programs using process management system calls

2. Experimental Procedure

   i. Analyse the problem statement

   ii. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code

   iii. Implement the algorithm in C language

   iv. Compile the C program

   v. Test the implemented program vi. Document the Results vii. Analyse and discuss the outcomes of your experiment

3. Questions

Name: Subhendu Maji Roll number: 18ETCS002121

Implement the following operations in C

Create 5 processes and distinguish parent and child processes. And also display process ID and its parent ID of all the created processes using process management system calls

4. Calculations/Computations/Algorithms

```
STEP 1: Start.

STEP 2: print output : getpid() , getppid()

STEP 3: for i = 0 to 4

        3.1 if fork() == 0

              3.1.1 print output : getpid() , getppid()

              3.1.2 exit(0)

STEP 4: Stop
```

5. Presentation of Results

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> // for system calls

int main()
{

    printf("child pid = %d from parent pid = %d \n", getpid(), getppid());
    // creating 4 process from 1 same parent
    for (int i = 0; i < 4; i++) // 4 processes
    {
        if (fork() == 0)
        {
            printf("child pid = %d from parent pid = %d \n", getpid(), getppid());
            exit(0);
        }
    }
    return (EXIT_SUCCESS);
}
```

Figure 1 Source Code

```
lab:
    @gcc lab2.c
run:
    @./a.out

clean:
    @rm *.out

.PHONY: clean
```

Figure 2 makefile

```
subhendu@LAPTOP-AL8CTHTV /mnt/d/RUAS-sem-05/OS/lab2 make lab
subhendu@LAPTOP-AL8CTHTV /mnt/d/RUAS-sem-05/OS/lab2 make run
child pid = 187 from parent pid = 186
child pid = 188 from parent pid = 187
child pid = 189 from parent pid = 187
child pid = 190 from parent pid = 187
child pid = 191 from parent pid = 187
subhendu@LAPTOP-AL8CTHTV /mnt/d/RUAS-sem-05/OS/lab2
```

Figure 3 execution

6. Analysis and Discussions

The `fork()` system call is a system call that is used to create process. This call creates a child process. The child process and the parent process, which called the `fork()` system call, execute the code that is present after the fork system called.

The parent and child processes do not share the same memory, but they can have multiple threads that share the same memory. The process run concurrently.

To distinguish the parent and the child, the value returned by the fork system call is used. For the parent process, the pid (process id) of the child is returned. For the child, the value returned is 0. If the value returned is negative, that means that there was an error that occurred while creating the child process.

The general intuition behind this program is that the parent processes execute the operations while the child operations call `fork()`.

7. Conclusions

A system call that creates a new process identical to the calling one – Makes a copy of text, data, stack, and heap – Starts executing on that new copy

Uses of `fork()` – To create a parallel program with multiple processes

8. Comments

1. Limitations of Experiments

The `fork()` system call creates processes. Each process consumes lot of system resources, such as CPU, memory, I/O etc. if used without care, it can lead to system overloads and other catastrophic failure.

2. Limitations of Results

The child and parent processes do not share the same memory space.

3. Learning happened

The method to create processes using the fork memory call was learned.