Name:- Kaushal Vashisth Roll no:- 18ETCS002147

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

   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

Algorithm:-

```
Step1:- start
Step2:-declare int pid,a,status
Step3:-pid<-fork()
Step4:-if(pid==0) do:
            4.1) pid<-fork()
            4.2) if(pid==0) do:
                  4.2.1) pid<-fork()
                  4.2.2) if(pid==0) do:
                        4.2.2.1) pid<-fork()
Step5:- write("\n process id: %d parent id: %d",getpid(),getppid())
Step6:-waitpid(-1,&status,0)
Step7:- stop
```

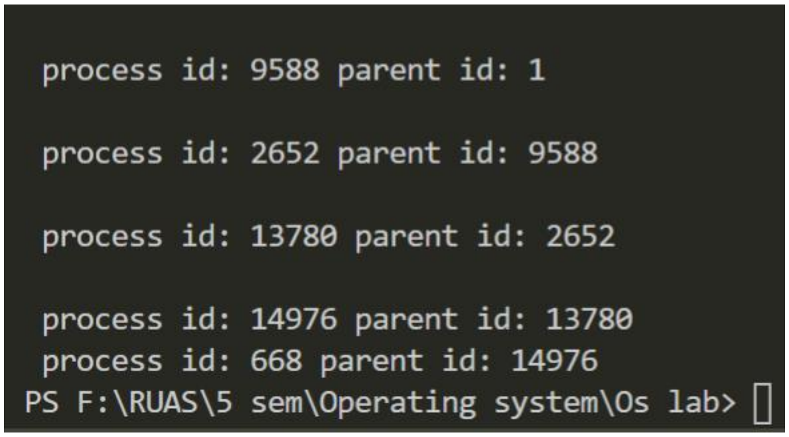
5. Presentation of Results

Program:-

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


 int main() {
    int pid,a,status;
    pid=fork();
    if(pid==0){
        pid=fork();
        if(pid==0){
            pid=fork();
            if(pid==0){
                pid==fork();
            }
        }
    }
    printf("\n process id: %d parent id: %d \n",getpid(),getppid());
    waitpid(-1,&status,0);
    return (EXIT_SUCCESS);
}
```

Output:-



```

 process id: 9588 parent id: 1

 process id: 2652 parent id: 9588

 process id: 13780 parent id: 2652

 process id: 14976 parent id: 13780
 process id: 668 parent id: 14976
PS F:\RUAS\5 sem\Operating system\Os lab> 
```

Figure 1:Output showing the child process id and parent process id.

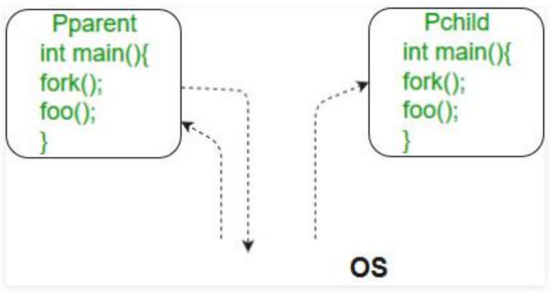
6. Analysis and Discussions

Fork system call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

7. Conclusions

It takes no parameters and returns an integer value. Below are different values returned by fork().

1. Negative Value: creation of a child process was unsuccessful.
2. Zero: Returned to the newly created child process.
3. Positive value: Returned to parent or caller. The value contains process ID of newly created child process.



`

8. Comments

1. Limitations of Experiments

   If the fork is 1 the process won't call the next function.

2. Limitations of Results

   The program can buffer causing the statement before fork to execute more than once.

3. Learning happened

   1. Fork() system call.
   2. How fork() works
   3. Getpid() and getppid()

4. Recommendations

   Using Wait function to avoid parent process to end without the child process being execute.