

Shri Ramdeobaba College of Engineering and Management, Nagpur



Department of Information Technology

Session: 2020-21

Name of Student : Malhar Deshkar

Roll No : 44

Semester : V

Course Name : Operating System Lab

Course Code : ITP351

Course Coordinator : Prof. Purushottam Jiwatram Assudani

Practical : 03

Aim : Write C programs using the following system calls of Linux operating system
Fork, wait, sleep, kill, exec, exit, getpid, getppid.

Theory:

fork():

All processes in Unix start their life via the **fork** system call. The **fork** function creates a copy of the calling process. The only distinction between the 2 processes is that the value returned to one of them (referred to as the parent process) will be the process ID of the copy. The copy is usually referred to as the child process. The value returned to the child process will be zero.

getpid():

The **getpid** function simply returns the process ID (PID) of the caller

getppid():

The **getppid** function returns the process ID of the caller's parent process.

wait():

The **wait** function suspends execution of the current process until a child has exited. If a child has already exited by the time of the call (a so-called "zombie" process), the function returns immediately. Any system resources used by the child are freed.

kill():

The **kill** function allows one process to send a signal to another process.

exec():

The **exec** family of functions overlay a new process image on an old process. The new process image is constructed from an ordinary, executable file. This file is either an executable object file, or a file of data for an interpreter. There can be no return from a successful **exec** because the calling process image is overlaid by the new process image.

exit():

exit() is a system call you enter once and never leave. It decrements the number of processes in the system by one.

exit() also accepts an exit status as a parameter, which the parent process can receive (or even has to receive), and which communicates the fate of the child to the parent.

sleep():

The function sleep gives a simple way to make the program wait for a short interval.

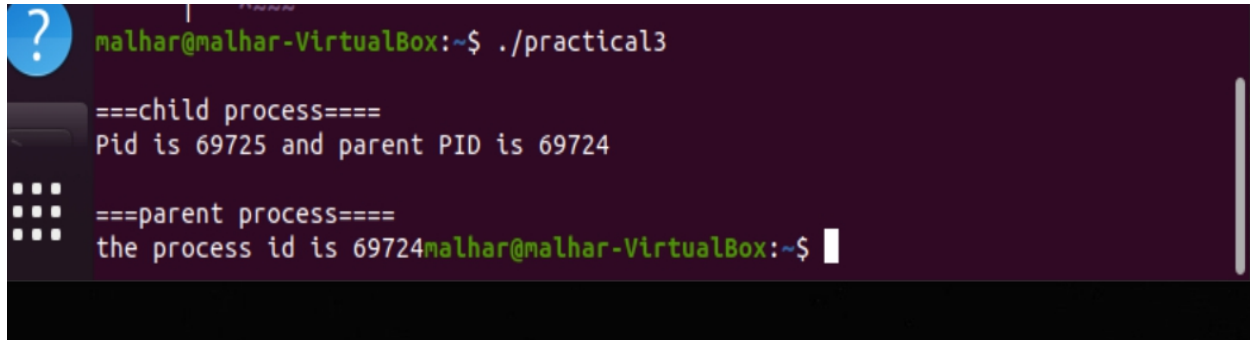
Code:

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

int main()
{
    int pid_t=0;
    int ret_value=0;
    ret_value = fork();
    if(ret_value<0)
    {
        printf("\n fork has failed\n");
    }

    else if(ret_value==0)
    {
        printf("\n====child process====\n");
        printf("Pid is %d and PID is %d\n",getpid(),getppid());
        sleep(10);
    }
    else
    {
        wait();
        printf("\n====parent process====\n");
        printf("the process id is %d", getpid());
        sleep(10);
        kill();
    }
    return 0;
}
```

Output:

A screenshot of a terminal window with a dark purple background. The prompt is 'malhar@malhar-VirtualBox:~\$'. The user has entered './practical3'. The output shows '===child process===', 'Pid is 69725 and parent PID is 69724', '===parent process===', and 'the process id is 69724'. The prompt 'malhar@malhar-VirtualBox:~\$' is visible again at the end of the output line. There is a blue circle with a white question mark on the left side of the terminal window.

```
malhar@malhar-VirtualBox:~$ ./practical3
===child process===
Pid is 69725 and parent PID is 69724
===parent process===
the process id is 69724malhar@malhar-VirtualBox:~$
```

Conclusion:

System calls of Linux operating system Fork, wait, sleep, kill, exec, exit, getpid, getppid were successfully implemented in the above program