ASSIGNMENT NUMBER: C3

TITLE : Study of UNIX system calls for process management.

PROBLEM STATEMENT:

Basics of process management and Linux environment.

OBJECTIVES:

- To get familiar with Linux programming
- To study basic Linux commands and utilites
- Learn process and thread management calls in Linux.

OUTCOMES:

The students will be able to

- Execute basic Linux commands.
- Make use of Linux system calls related to process management.
- Implement and execute programs in Linux environment.

THEORY:

- **fork** - create a child process

```
#include <sys/types.h>
    #include <unistd.h>

    pid_t fork(void);
```

fork() creates a new process by duplicating the calling process. The new process is referred to as the *child* process. The calling process is referred to as the *parent* process.

The child process is an exact duplicate of the parent process except for the following points:

* The child has its own unique process ID, and this PID does not match the ID of any existing process group or session.

* The child's parent process ID is the same as the parent's process ID.

RETURN VALUE

On success, the PID of the child process is returned in the parent, and 0 is returned in the child. On failure, -1 is returned in the parent, no child process is created.

- **An exec** call will load a *new* program into the process and replace the current running program with the one specified. For example, consider this program, which will execute the ls -l command in the current directory:

There are three main versions of exec which we will focus on:

- execv(char * path, char * argv[]) : given the path to the program and an argument array, load and execute the program

- execvp(char * file, char * argv[]) : given a file(name) of the program and an argument array, find the file in the environment PATHand execute the program
- execvpe(char * file, char * argv[], char * envp[]) given a file(name), an argument array, and the enviroment settings, within the enviroment, search the PATH for the program named file and execute with the arguments.
- **Waiting on a child with wait()**

The wait() system call is used by a parent process to *wait* for the status of the child to change. A status change can occur for a number of reasons, the program stopped or continued, but we'll only concern ourselves with the most common status change: the program terminated or exited. (We will discuss stopped and continued in later lessons.)

System calls provide the interface between a process and the operating system. *These system calls are the routine services of the operating system*.
Linux system call fork () creates a process Exec() ,join() etc.
Steps To Do/algorithm:

1. Study the various Linux process handling system calls.
2. Execute basic Linux commands.
3. Print the information about a process its task structure ids etc.

C**ONCLUSION :** We have successfully implemented different UNIX calls for process management by performing this experiment.