	SHRI SANT GAJANAN MAHARAJ COLLEGE ENGG. SHEGAON		LABORATORY MANUAL
	PRACTICAL EXPERIMENT INSTRUCTION SHEET		
	EXPERIMENT TITLE: **Introduction to Linux OS**		
EXPERIMENT NO.: SSGMCE/WI/ASH/01/1A6/01		ISSUE NO.:	**ISSUE DATE:**26.08.23
REV. DATE: 26.08.23	REV. NO.: 00	DEPTT. : INFORMATION TECHNOLOGY.	
LABORATORY: OPERATING SYSTEMS		SEMESTER : IV	PAGE NO. 01 OF 05

Date:

INTRODUCTION TO LINUX OS

01. AIM :

Write a program to create a new process.

02. FACILITIES :

Linux OS: To run the program since `fork()` is a system call specific to Unix-like systems.
C Compiler: GCC or any other C compiler to compile the code.
Text Editor: To write the program (e.g., Nano, Vim).
Terminal: To execute the compiled program and see the output.

03. SCOPE :

Process Creation: Understand how a new process is created using the `fork()` system call.
Parent and Child Process: Learn the interaction between the parent and child process.
Process Management: Gain insight into process IDs (PID) and parent process IDs (PPID) in Linux.

04 THEORY:

Processes in Linux/UNIX:

A program/command when executed, a special instance is provided by the system to the process. This instance consists of all the services/resources that may be utilized by the process under execution.

- Whenever a command is issued in Unix/Linux, it creates/starts a new process. For example, pwd when issued which is used to list the current directory location the user is in, a process starts.
- Through a 5 digit ID number Unix/Linux keeps an account of the processes, this number is called process ID or PID. Each process in the system has a unique PID.
- Used up pid's can be used in again for a newer process since all the possible combinations are used.
- At any point of time, no two processes with the same pid exist in the system because it is the pid that Unix uses to track each process.

`fork()` System Call:

In Linux, the **`fork()`** system call is essential for process creation. When a process calls `fork()`, it creates a duplicate of itself, known as the **child process**. The child process is almost identical to the parent process, inheriting the same code, data, and resources. However, they are distinct processes with separate memory spaces, allowing each to operate independently.

The `fork()` system call works by creating a new process by duplicating the parent process's context. After a successful `fork()`, both the parent and child processes execute the same program, but they can diverge based on the return value of `fork()`:

- If **`pid < 0`**, it indicates that the `fork()` operation failed. This might happen due to system resource limitations, and no child process is created.
- If **`pid == 0`**, the process is in the **child process**. This means the new process that was created by `fork()` will follow this path.
- If **`pid > 0`**, the process is in the **parent process**. The value of `pid` will be the Process ID (PID) of the newly created child process.

Even though the child process is a duplicate of the parent process at the time of creation, the kernel assigns a unique PID to the child, which differentiates it from the parent. The **child process** can execute its own code independently from the parent process after forking.

The `fork()` system call plays a critical role in the design of multitasking operating systems, allowing a process to spawn multiple child processes to perform concurrent tasks. In some cases, the child process can execute entirely different tasks from the parent, making this system call powerful for process management in operating systems.

In this experiment, we use the `getpid()` and `getppid()` functions:

- **`getpid()`** returns the process ID (PID) of the calling process. It is used to identify the process in question.
- **`getppid()`** returns the parent process ID (PPID) of the calling process, which tells the child who its parent is.

Key Concepts:

1. **Process Creation**:
   - The **`fork()`** system call is responsible for creating new processes in a Linux-based operating system. This is a critical operation in process management as it allows a running process (parent) to spawn new processes (children), which can execute concurrently.
2. **Parent Process**:

- The **parent process** is the process that initiates the `fork()` system call. It continues to execute after the `fork()` but has the child process running in parallel. The parent process can communicate or synchronize with the child process as needed, making the concept of parent-child process interaction central to process management.

3. **Child Process**:
   - The **child process** is a copy of the parent process that is created when the `fork()` system call is invoked. After forking, the child process has its own unique PID, but it shares many characteristics with the parent, such as the code and memory layout, although they operate independently.
4. **PID (Process ID)**:
   - The **PID** is a unique identifier assigned to every process in the system. It helps the operating system keep track of running processes. The PID is assigned when the process is created, and it is used by the system to manage and control the process lifecycle.
5. **PPID (Parent Process ID)**:
   - The **PPID** refers to the Parent Process ID. It is the PID of the process that created (or forked) the current process. For a child process, the PPID is the PID of the parent process. This relationship is crucial because it shows the hierarchical structure of processes in a system.

05 PROGRAM:

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

int main() {
  pid_t pid;

  // Create a new process
  pid = fork();

  if (pid < 0) {
    // Fork failed
    printf("Fork Failed!\n");
    return 1;
  } else if (pid == 0) {
      // Child process
      printf("This is the child process. PID: %d, Parent PID: %d\n", getpid(), getppid());
    } else {
      // Parent Process
      printf("This is the parent process. PID: %d, Child PID: %d\n", getpid(), pid);
    }

  return 0;
}
```

08 OUTPUT:

```
This is the parent process. PID: 1234, Child PID: 1235
This is the child process. PID: 1235, Parent PID: 1234
```

08 CONCLUSION:

The program successfully demonstrates the use of the `fork()` system call in Linux to create a new process. It highlights the creation of a parent-child relationship, where both processes execute independently with their own process IDs. This experiment helps in understanding how processes are created and managed in Unix-like systems, laying the foundation for more advanced concepts in process management.

09 VIVA QUESTIONS:

PREPARED BY **H. P. Amle**	**APPROVED BY** **HOD**