**Ex. No: 2**

**Date:**

**SYSTEM CALLS PROGRAMMING**

**Problem Statement:**

Create a simple Python program that uses system calls for process management. The program should demonstrate the use of fork(), getpid(), getppid(), sleep(), exit().

**Problem Description:**

This Python program illustrates process management using system calls like fork(), getpid(), getppid(), sleep(), and exit(). It initiates with displaying the parent process's PID, then creates a child process, demonstrating both the child's PID and a simulated delay. Following this, the child process exits gracefully. Meanwhile, the parent process waits for the child process to complete and displays the child's exit status, offering a succinct demonstration of process control.

**Synopsis:**

1. **fork():** Creates a new process by duplicating the current one.
2. **getpid():** Retrieves the Process ID (PID) of the current process.
3. **getppid():** Retrieves the PID of the parent process.
4. **sleep():** Delays process execution for a specified time.
5. **exit():** Terminates the process, providing an exit status.

**Code 2a:**

import os

import time

def child\_process():

    print("Child Process - PID:", os.getpid())

    print("Child Process - Parent PID:", os.getppid())

    time.sleep(2)

    print("Child Process - Exiting")

    os.\_exit(0)

def main():

    print("Parent Process - PID:", os.getpid())

    print("Parent Process - Forking a Child Process...")

    child\_pid = os.fork()

    if child\_pid == 0:

        child\_process()

    else:

        print("Parent Process - Waiting for the child process to complete...")

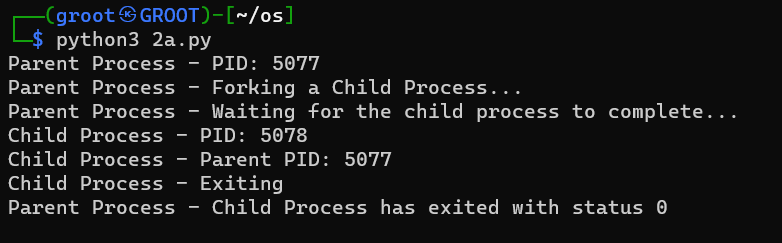
        \_, status = os.wait()

        print("Parent Process - Child Process has exited with status", status)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**

****

**Problem Statement:**

Create a simple Python program that uses system calls for file operations. The program should demonstrate the use of read(), write(), and close() system calls.

**Problem Description:**

This Python program highlights file operations using system calls like read(), write(), and close(). It starts by opening a file for read and write access, then writes a sample text to the file and repositions the file cursor. Subsequently, it reads the data from the file, prints it to the console, and closes the file. This concise program effectively showcases the core file operation system calls.

1. **Synopsis:**
2. **read():** Reads data from a file descriptor.
3. **write():** Writes data to a file descriptor.
4. **close():** Closes a file descriptor, releasing associated resources.

**Code 2b:**

import os

def file\_operations(filename):

    try:

        fd = os.open(filename, os.O\_RDWR | os.O\_CREAT)

        if fd:

            os.write(fd, b"This is a sample text.")

            os.lseek(fd, 0, os.SEEK\_SET)

            data = os.read(fd, 1024)

            print("Read data:", data.decode())

            os.close(fd)

        else:

            print("Error opening file")

    except OSError as e:

        print("File operation error:", e)

def main():

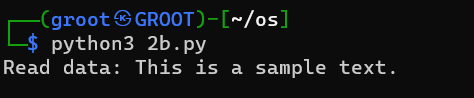
    filename = "sample.txt"

    file\_operations(filename)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**

****

**Result:**

Thus, system calls for process management and file management were executed successfully