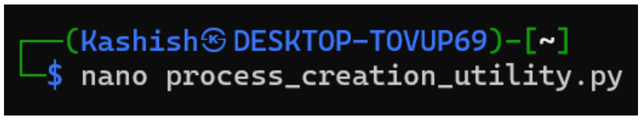
Operation System

Lab Assignment 1

Task 1: Process Creation Utility

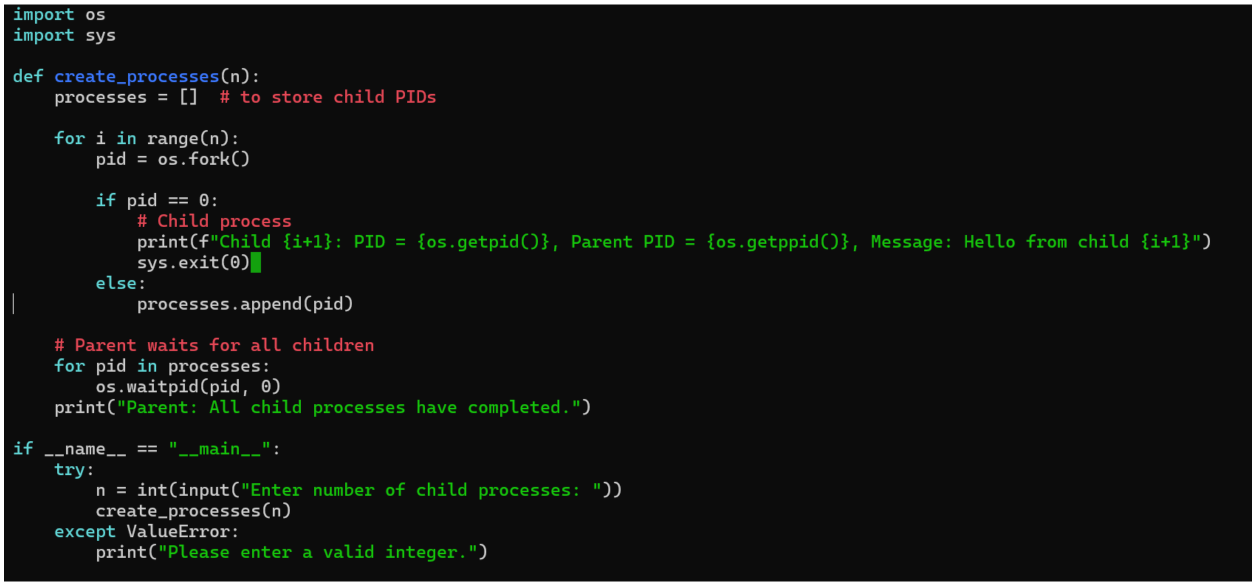
Write a Python program that creates N child processes using os.fork(). Each child prints:
- Its PID
- Its Parent PID
- A custom message
The parent should wait for all children using os.wait().

1. Creating a file in nano editor:



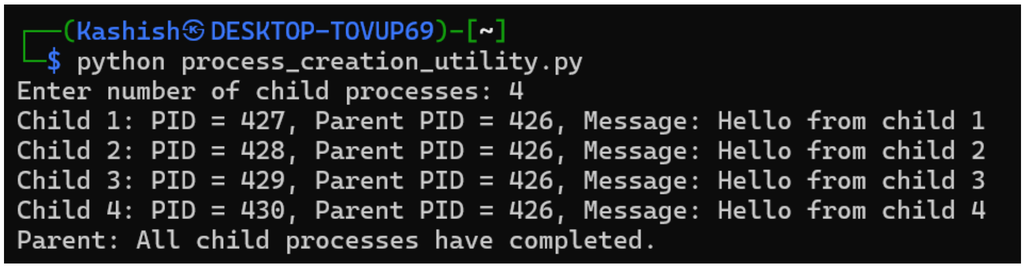
```
┌──(Kashish㉿DESKTOP-TOVUP69)-[~]
└─$ nano process_creation_utility.py
```

2. Write python script



```
import os
import sys

def create_processes(n):
    processes = []  # to store child PIDs

    for i in range(n):
        pid = os.fork()

        if pid == 0:
            # Child process
            print(f"Child {i+1}: PID = {os.getpid()}, Parent PID = {os.getppid()}, Message: Hello from child {i+1}")
            sys.exit(0)
        else:
            processes.append(pid)

    # Parent waits for all children
    for pid in processes:
        os.waitpid(pid, 0)
    print("Parent: All child processes have completed.")

if __name__ == "__main__":
    try:
        n = int(input("Enter number of child processes: "))
        create_processes(n)
    except ValueError:
        print("Please enter a valid integer.")
```

3. Executing python file

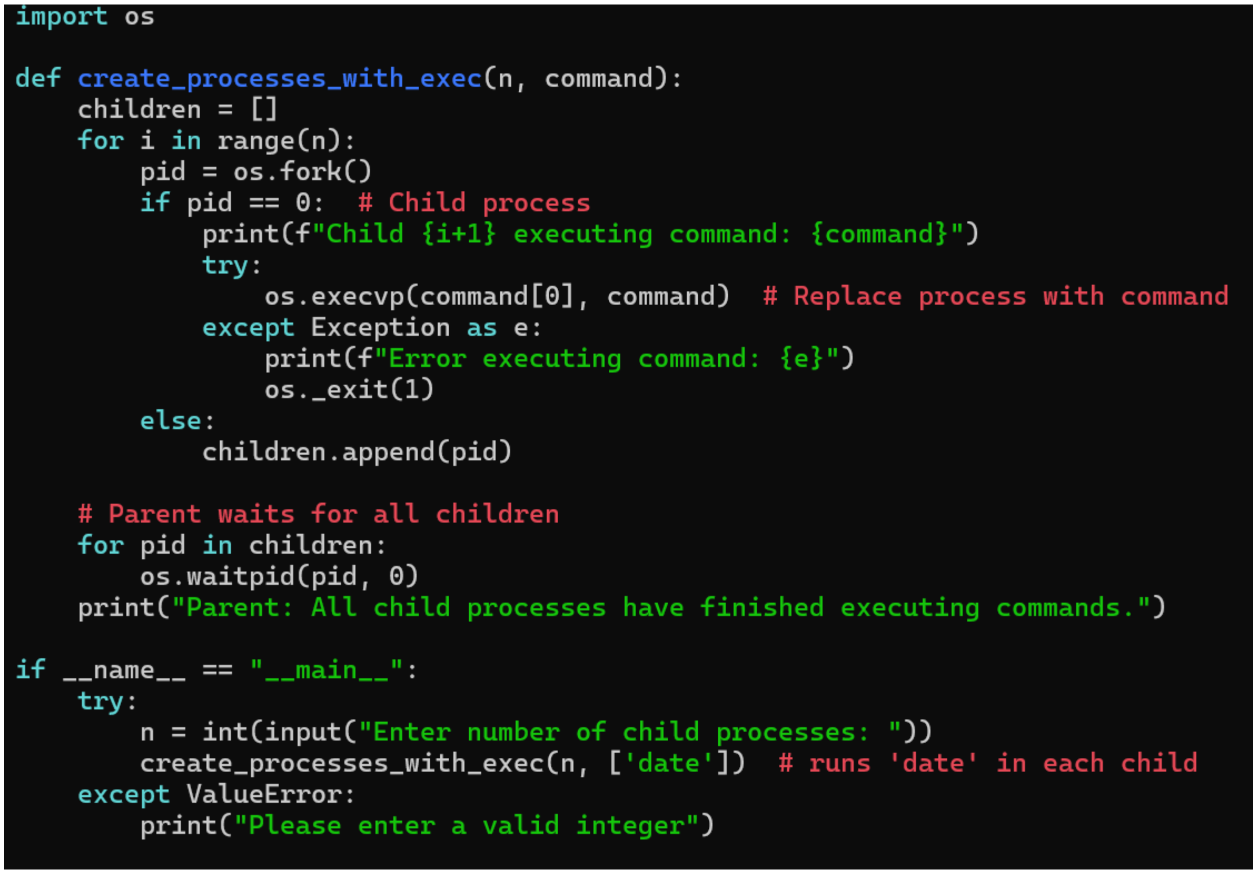


```
┌──(Kashish㉿DESKTOP-TOVUP69)-[~]
└─$ python process_creation_utility.py
Enter number of child processes: 4
Child 1: PID = 427, Parent PID = 426, Message: Hello from child 1
Child 2: PID = 428, Parent PID = 426, Message: Hello from child 2
Child 3: PID = 429, Parent PID = 426, Message: Hello from child 3
Child 4: PID = 430, Parent PID = 426, Message: Hello from child 4
Parent: All child processes have completed.
```

Task 2: Command Execution Using exec()

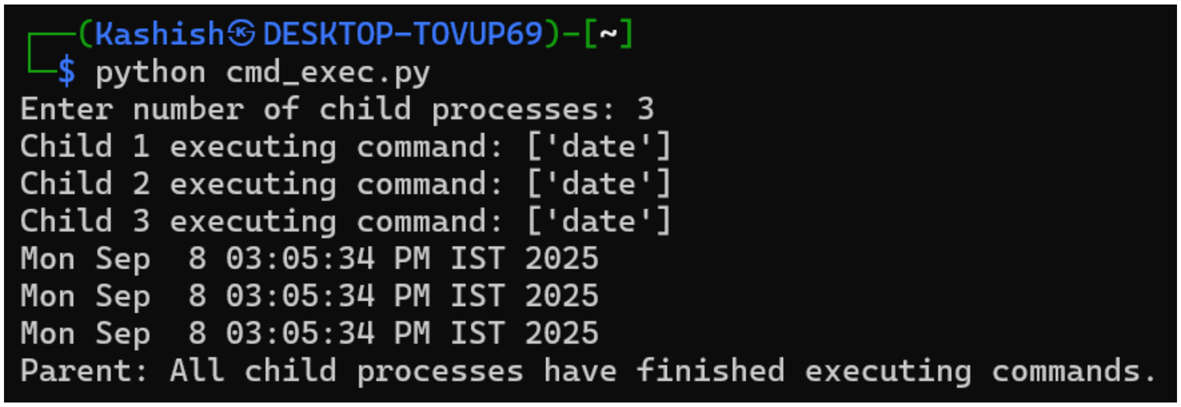
Modify Task 1 so that each child process executes a Linux command (ls, date, ps, etc.) using os.execvp() or subprocess.run().

- **Create file and write the python script**



```python
import os

def create_processes_with_exec(n, command):
    children = []
    for i in range(n):
        pid = os.fork()
        if pid == 0:  # Child process
            print(f"Child {i+1} executing command: {command}")
            try:
                os.execvp(command[0], command)  # Replace process with command
            except Exception as e:
                print(f"Error executing command: {e}")
                os._exit(1)
        else:
            children.append(pid)

    # Parent waits for all children
    for pid in children:
        os.waitpid(pid, 0)
    print("Parent: All child processes have finished executing commands.")

if __name__ == "__main__":
    try:
        n = int(input("Enter number of child processes: "))
        create_processes_with_exec(n, ['date'])  # runs 'date' in each child
    except ValueError:
        print("Please enter a valid integer")
```

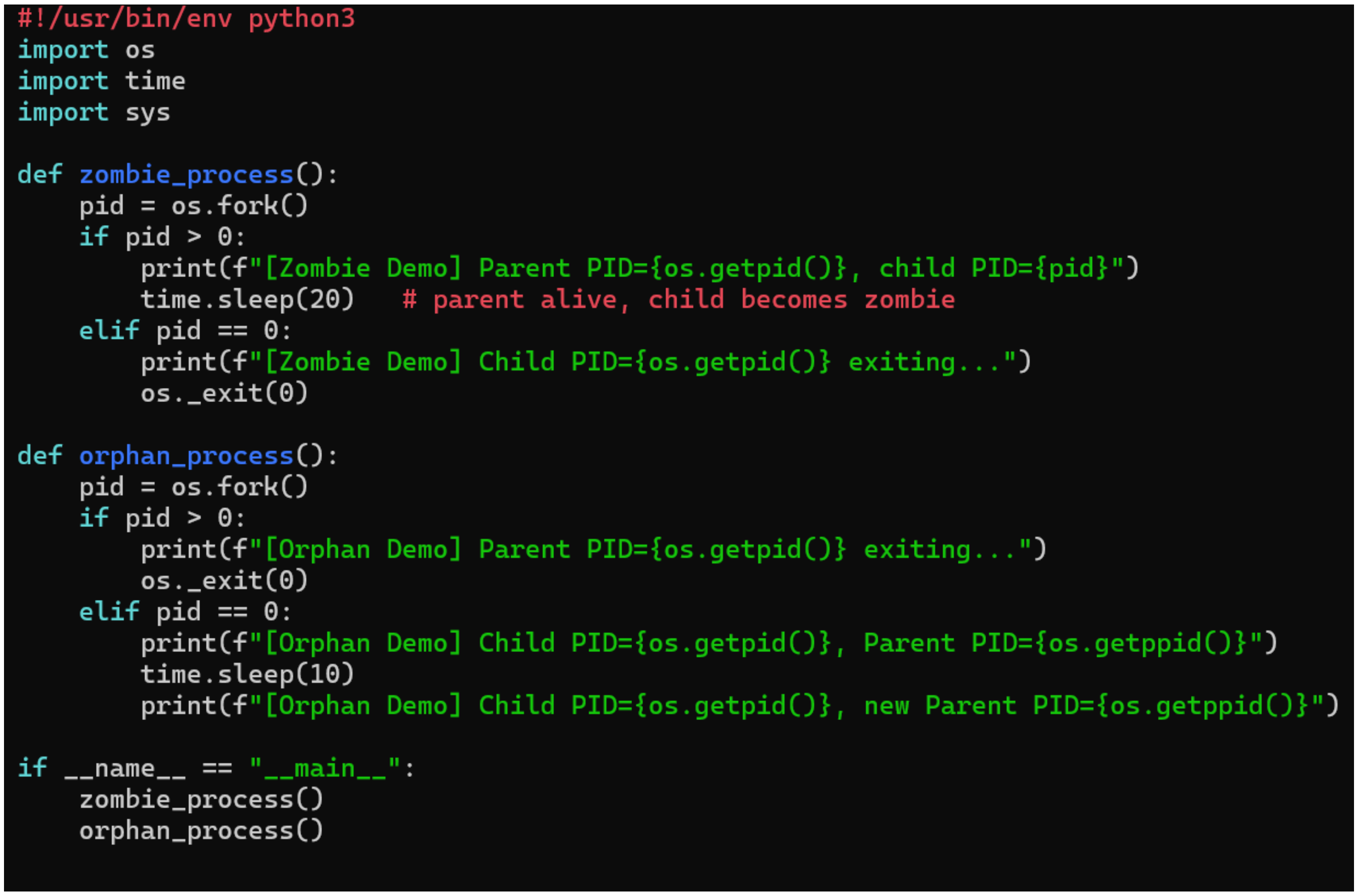
- **Output:**



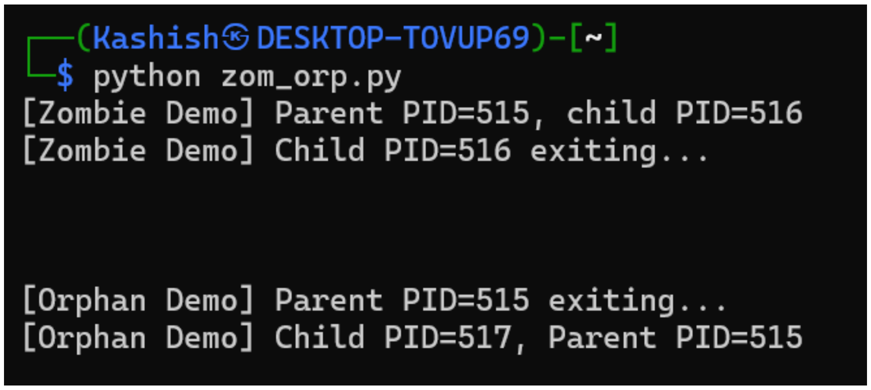
```
┌──(Kashish㉿DESKTOP-TOVUP69)-[~]
└─$ python cmd_exec.py
Enter number of child processes: 3
Child 1 executing command: ['date']
Child 2 executing command: ['date']
Child 3 executing command: ['date']
Mon Sep  8 03:05:34 PM IST 2025
Mon Sep  8 03:05:34 PM IST 2025
Mon Sep  8 03:05:34 PM IST 2025
Parent: All child processes have finished executing commands.
```

Task 3: Zombie & Orphan Processes

Zombie: Fork a child and skip wait() in the parent.
Orphan: Parent exits before the child finishes.
Use ps -el | grep defunct to identify zombies.



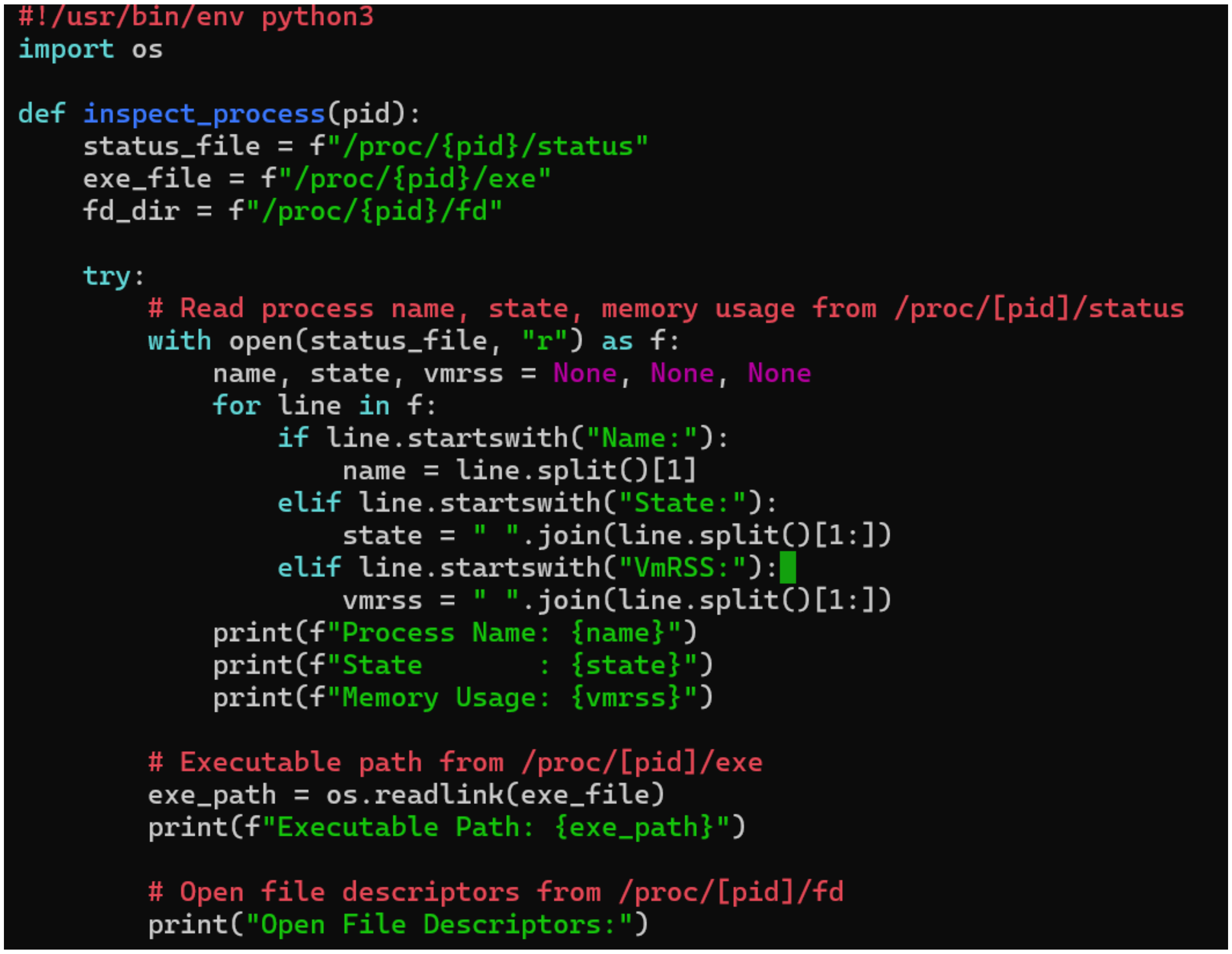
```python
#!/usr/bin/env python3
import os
import time
import sys

def zombie_process():
    pid = os.fork()
    if pid > 0:
        print(f"[Zombie Demo] Parent PID={os.getpid()}, child PID={pid}")
        time.sleep(20)   # parent alive, child becomes zombie
    elif pid == 0:
        print(f"[Zombie Demo] Child PID={os.getpid()} exiting...")
        os._exit(0)

def orphan_process():
    pid = os.fork()
    if pid > 0:
        print(f"[Orphan Demo] Parent PID={os.getpid()} exiting...")
        os._exit(0)
    elif pid == 0:
        print(f"[Orphan Demo] Child PID={os.getpid()}, Parent PID={os.getppid()}")
        time.sleep(10)
        print(f"[Orphan Demo] Child PID={os.getpid()}, new Parent PID={os.getppid()}")

if __name__ == "__main__":
    zombie_process()
    orphan_process()
```



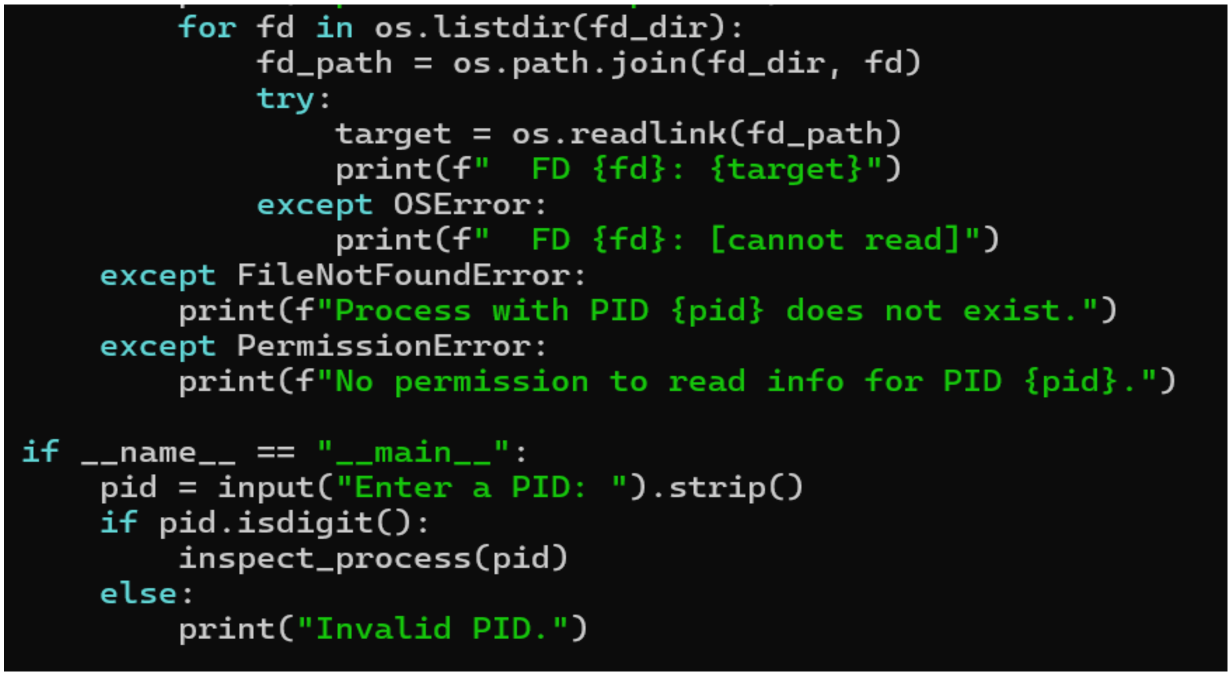
```
┌──(Kashish㉿DESKTOP-TOVUP69)-[~]
└─$ python zom_orp.py
[Zombie Demo] Parent PID=515, child PID=516
[Zombie Demo] Child PID=516 exiting...



[Orphan Demo] Parent PID=515 exiting...
[Orphan Demo] Child PID=517, Parent PID=515
```

Task 4: Inspecting Process Info from /proc

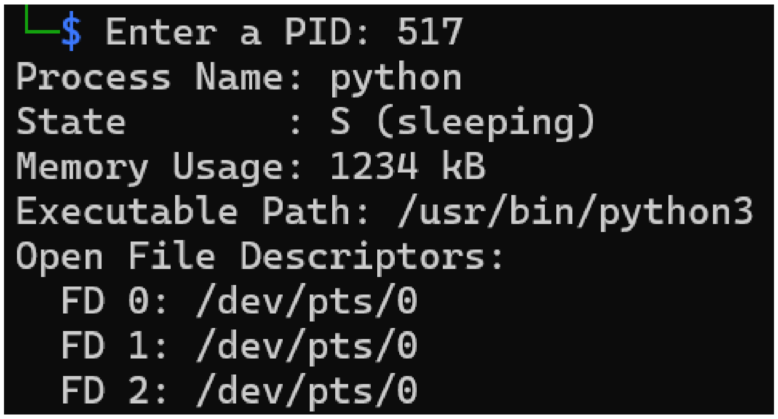
Take a PID as input. Read and print:
- Process name, state, memory usage from /proc/[pid]/status
- Executable path from /proc/[pid]/exe
- Open file descriptors from /proc/[pid]/fd



```python
#!/usr/bin/env python3
import os

def inspect_process(pid):
    status_file = f"/proc/{pid}/status"
    exe_file = f"/proc/{pid}/exe"
    fd_dir = f"/proc/{pid}/fd"

    try:
        # Read process name, state, memory usage from /proc/[pid]/status
        with open(status_file, "r") as f:
            name, state, vmrss = None, None, None
            for line in f:
                if line.startswith("Name:"):
                    name = line.split()[1]
                elif line.startswith("State:"):
                    state = " ".join(line.split()[1:])
                elif line.startswith("VmRSS:"):
                    vmrss = " ".join(line.split()[1:])
            print(f"Process Name: {name}")
            print(f"State       : {state}")
            print(f"Memory Usage: {vmrss}")

        # Executable path from /proc/[pid]/exe
        exe_path = os.readlink(exe_file)
        print(f"Executable Path: {exe_path}")

        # Open file descriptors from /proc/[pid]/fd
        print("Open File Descriptors:")
```



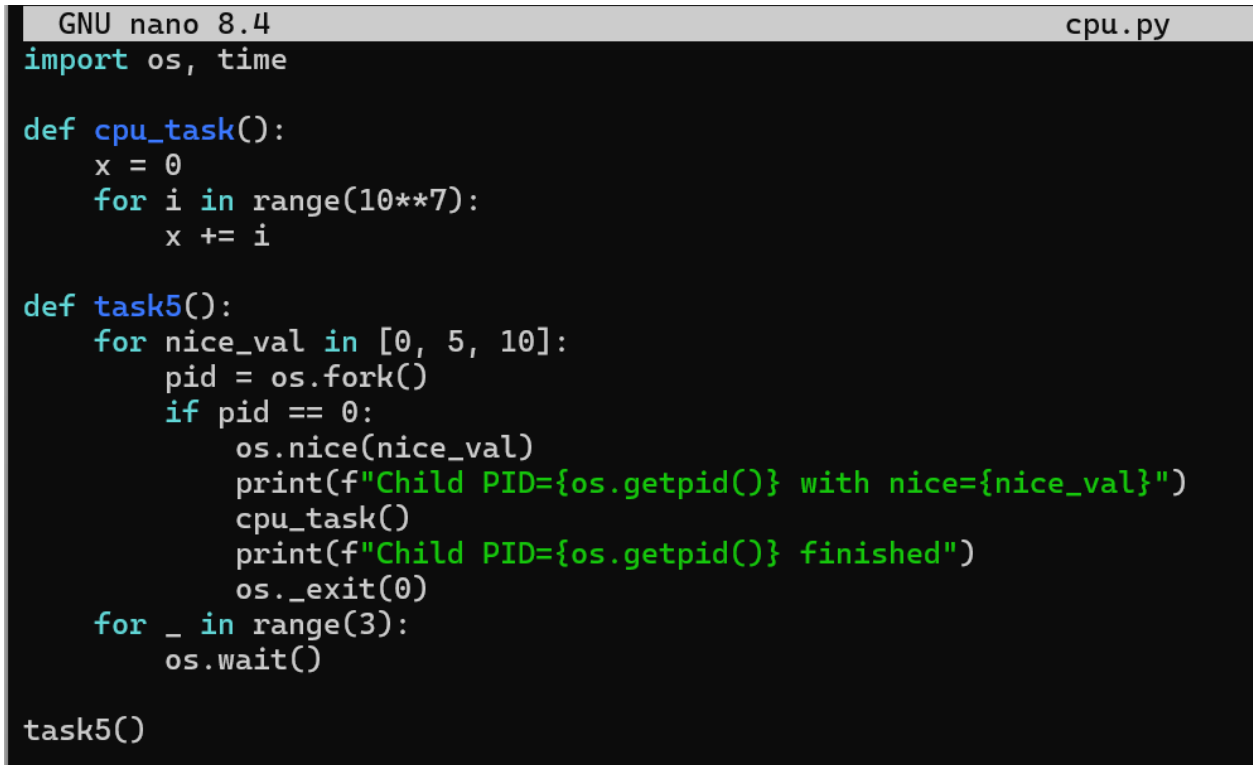
```
        for fd in os.listdir(fd_dir):
            fd_path = os.path.join(fd_dir, fd)
            try:
                target = os.readlink(fd_path)
                print(f"  FD {fd}: {target}")
            except OSError:
                print(f"  FD {fd}: [cannot read]")
    except FileNotFoundError:
        print(f"Process with PID {pid} does not exist.")
    except PermissionError:
        print(f"No permission to read info for PID {pid}.")

if __name__ == "__main__":
    pid = input("Enter a PID: ").strip()
    if pid.isdigit():
        inspect_process(pid)
    else:
        print("Invalid PID.")
```



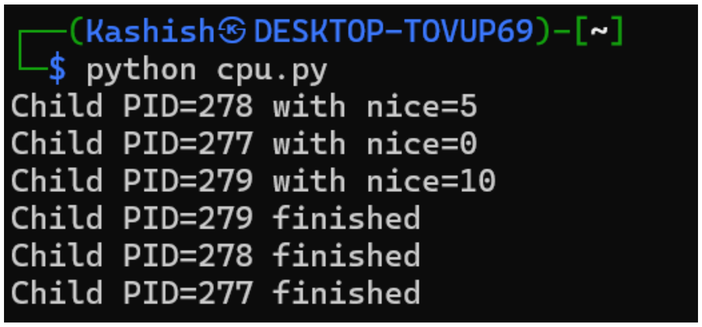
```
└─$ Enter a PID: 517
Process Name: python
State       : S (sleeping)
Memory Usage: 1234 kB
Executable Path: /usr/bin/python3
Open File Descriptors:
  FD 0: /dev/pts/0
  FD 1: /dev/pts/0
  FD 2: /dev/pts/0
```

Task 5: Process Prioritization

Create multiple CPU-intensive child processes. Assign different nice() values. Observe and log execution order to show scheduler impact.



```
  GNU nano 8.4                                                   cpu.py
import os, time

def cpu_task():
    x = 0
    for i in range(10**7):
        x += i

def task5():
    for nice_val in [0, 5, 10]:
        pid = os.fork()
        if pid == 0:
            os.nice(nice_val)
            print(f"Child PID={os.getpid()} with nice={nice_val}")
            cpu_task()
            print(f"Child PID={os.getpid()} finished")
            os._exit(0)
    for _ in range(3):
        os.wait()

task5()
```



```
┌──(Kashish㉿DESKTOP-TOVUP69)-[~]
└─$ python cpu.py
Child PID=278 with nice=5
Child PID=277 with nice=0
Child PID=279 with nice=10
Child PID=279 finished
Child PID=278 finished
Child PID=277 finished
```