LAB ASSIGNMENT-01

Experiment Title: Process Creation and Management Using Python OS Module

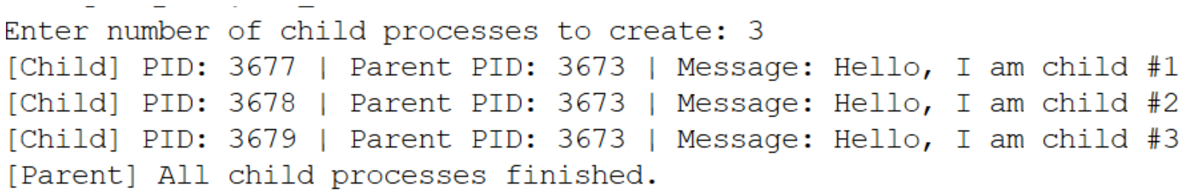
Task 1: Process Creation Utility

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().

Implementation:

```
import os

def create_processes(N):

    children = []

    for i in range(N):

        pid = os.fork()

        if pid == 0:      # Child process

            print(f"[Child] PID: {os.getpid()} | Parent PID: {os.getppid()} | Message: Hello, I am child #{i+1}")

            os._exit(0)   # Ensures child does not continue loop

        else:

            children.append(pid)

    # Parent waits for all children

    for cpid in children:

        os.waitpid(cpid, 0)

    print("[Parent] All child processes finished.")

if __name__ == "__main__":

    N = int(input("Enter number of child processes to create: "))

    create_processes(N)
```

Output:



```
Enter number of child processes to create: 3
[Child] PID: 3677 | Parent PID: 3673 | Message: Hello, I am child #1
[Child] PID: 3678 | Parent PID: 3673 | Message: Hello, I am child #2
[Child] PID: 3679 | Parent PID: 3673 | Message: Hello, I am child #3
[Parent] All child processes finished.
```

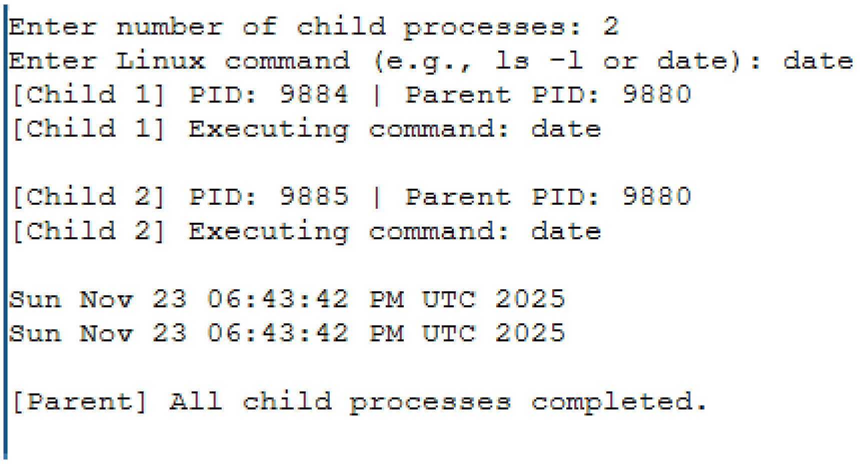
Task 2: Command Execution Using exec()

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

Implementation:

```
import os

def create_processes_with_commands(N, command):

  children = []

  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()}")

      print(f"[Child {i+1}] Executing command: {' '.join(command)}\n")

      os.execvp(command[0], command)

      # Note: No code after execvp() will run if successful

    else:

      children.append(pid)

  # Parent waits for all children

  for cpid in children:

    os.waitpid(cpid, 0)

  print("\n[Parent] All child processes completed.")

if __name__ == "__main__":

  N = int(input("Enter number of child processes: "))

  cmd_input = input("Enter Linux command (e.g., ls -l or date): ").split()

  create_processes_with_commands(N, cmd_input)
```

Output:



```
Enter number of child processes: 2
Enter Linux command (e.g., ls -l or date): date
[Child 1] PID: 9884 | Parent PID: 9880
[Child 1] Executing command: date

[Child 2] PID: 9885 | Parent PID: 9880
[Child 2] Executing command: date

Sun Nov 23 06:43:42 PM UTC 2025
Sun Nov 23 06:43:42 PM UTC 2025

[Parent] All child processes completed.
```

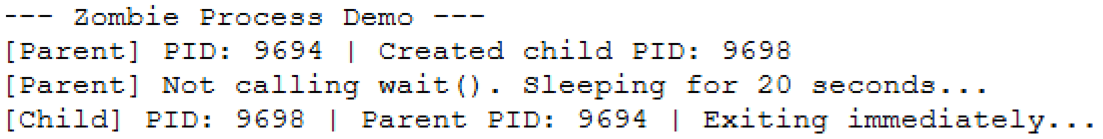
Task 3: Zombie & Orphan Processes

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.

Implementation:

Zombie Process:

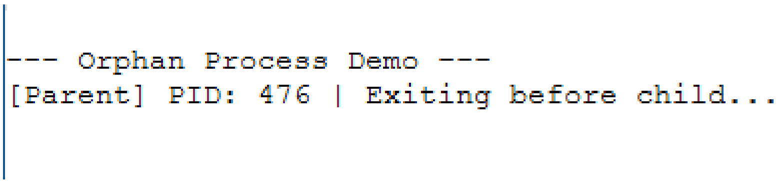
```
import os

import time

print("\n--- Zombie Process Demo ---")

pid = os.fork()

if pid == 0:

    print(f"[Child] PID: {os.getpid()} | Parent PID: {os.getppid()} | Exiting immediately...")

    os._exit(0)   # Child finishes

else:

    print(f"[Parent] PID: {os.getpid()} | Created child PID: {pid}")

    print("[Parent] Not calling wait(). Sleeping for 20 seconds...")

    time.sleep(
```



```
--- Zombie Process Demo ---
[Parent] PID: 9694 | Created child PID: 9698
[Parent] Not calling wait(). Sleeping for 20 seconds...
[Child] PID: 9698 | Parent PID: 9694 | Exiting immediately...
```

Orphan Process:

```
import os

import time

print("\n--- Orphan Process Demo ---")

pid = os.fork()

if pid == 0:

    time.sleep(10)

    print(f"[Child] I am now orphaned. PID: {os.getpid()} | New Parent PID (should be 1): {os.getppid()}")

else:

    print(f"[Parent] PID: {os.getpid()} | Exiting before child...")

    os._exit(0)   # Parent exits immediately
```



```
--- Orphan Process Demo ---
[Parent] PID: 476 | Exiting before child...
```

Task 4: Inspecting Process Info from /proc

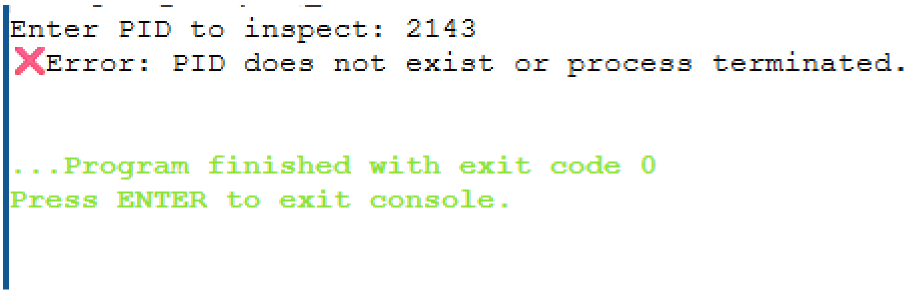
Take a PID as input. Read and print:
- Process name, state, memory usage from /proc/[pid]/status
- Executable path from /proc/[pid]/exe
- Open file descriptors from /proc/[pid]/fd

Implementation:

```
import os

def inspect_process(pid):

    status_file = f"/proc/{pid}/status"

    exe_file = f"/proc/{pid}/exe"

    fd_dir = f"/proc/{pid}/fd"

    # --- Read basic fields from /proc/[pid]/status ---

    try:

        with open(status_file, "r") as f:

            name = state = memory = None
            for line in f:
              if line.startswith("Name:"):
                name = line.split(":")[1].strip()
              elif line.startswith("State:"):
                state = line.split(":")[1].strip()
              elif line.startswith("VmRSS:"):
                memory = line.split(":")[1].strip()   # Resident memory
    except FileNotFoundError:
        print("❌ Error: PID does not exist or process terminated.")
        return
    print("\n--- Process Information ---")
    print(f"PID: {pid}")
    print(f"Name: {name}")
    print(f"State: {state}")
    print(f"Memory Usage (Resident): {memory if memory else 'N/A'}")
    # --- Executable path from /proc/[pid]/exe ---
    try:
        exe_path = os.readlink(exe_file)
        print(f"Executable Path: {exe_path}")
    except Exception:
        print("Executable Path: Unable to read (permission denied or missing)")
    # --- List open file descriptors /proc/[pid]/fd ---
    print("\n--- Open File Descriptors ---")
    try:
        fds = os.listdir(fd_dir)
        if not fds:
            print("(None)")
        else:
            for fd in fds:
              try:
                target = os.readlink(os.path.join(fd_dir, fd))
```

```
                    print(f"FD {fd} -> {target}")
                except Exception:
                    print(f"FD {fd} -> (unreadable)")
    except Exception:
        print("Unable to list FDs (permission denied or missing)")
if __name__ == "__main__":
    pid = input("Enter PID to inspect: ").strip()
    inspect_process(pid)
```

Output:



```
Enter PID to inspect: 2143
❌Error: PID does not exist or process terminated.


...Program finished with exit code 0
Press ENTER to exit console.
```

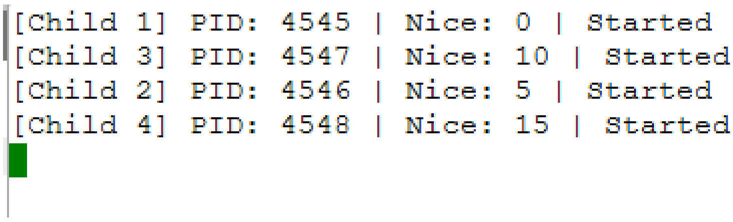
Task 5: Process Prioritization

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

Implementation:

```
import os
import time
def cpu_task(task_id, nice_val):
    # Change process priority
    os.nice(nice_val)
    start = time.time()
    print(f"[Child {task_id}] PID: {os.getpid()} | Nice: {nice_val} | Started")
    # CPU-intensive workload (dummy loop)
    total = 0
    for i in range(90_000_000):
        total += i*i
    end = time.time()
    duration = end - start
    print(f"[Child {task_id}] Finished | Nice: {nice_val} | Time: {duration:.2f} sec")
def run_priority_demo():
    # Define nice levels for each child
    nice_values = [0, 5, 10, 15]   # lower = higher priority
    children = []
    for idx, nval in enumerate(nice_values, start=1):
        pid = os.fork()
        if pid == 0:  # Child
            cpu_task(idx, nval)
            os._exit(0)   # Prevent child from continuing loop
        else:
            children.append(pid)
    # Parent waits for all
    for cpid in children:
        os.waitpid(cpid, 0)
    print("\n[Parent] All CPU tasks completed.")
if __name__ == "__main__":
    run_priority_demo()
```

Output:



```
[Child 1] PID: 4545 | Nice: 0 | Started
[Child 3] PID: 4547 | Nice: 10 | Started
[Child 2] PID: 4546 | Nice: 5 | Started
[Child 4] PID: 4548 | Nice: 15 | Started
```