# Lab Experiment Sheet-1

School of Engineering and Technology
Course Code & Name: ENCS351 Operating System
Program Name: B.Tech CSE, AI ML, Data Science, Cyber, FSD, UX/UI

**Name – Yatharth Chopra** **Course – BTECH CSE DS -Sem 5**

**Roll No -2301420022**

# Summary of objectives

**Task 1: Process Creation Utility**

To accomplish this, I wrote a Python function called task1. I used a

for loop to call **os.fork()** three times, which created three distinct child processes. Inside the code block for each child (where

pid == 0), I used os.getpid() and os.getppid() to print its own Process ID and its parent's ID, along with a custom message. In the parent's code block, I made sure the parent process wouldn't exit prematurely by calling

**os.waitpid()** for each child, ensuring it waited for all of them to finish their execution.

---

**Task 2: Command Execution Using exec()**

For this task, I created a function task2 that forked a single child process. In the section of code executed by the child, I used

**os.execvp("ls", ["ls", "-l"])**. This system call replaced the child process's own code with the

ls -l command, effectively making the child execute that command in the terminal. The parent process simply waited for the command to finish before the script continued.

---

**Task 3: Zombie & Orphan Processes**

I simulated these two special process states in separate functions.

- **Zombie Process**: I created a child that printed a message and exited immediately using os._exit(0). The key to creating a zombie was making the parent process

**skip the os.wait() call**. Instead, I made the parent sleep for 10 seconds. During this time, the child was "defunct" or a zombie because it had terminated, but the parent hadn't yet acknowledged its termination to clean it up from the process table.

- **Orphan Process**: I did the reverse for the orphan process. I made the

**parent process exit immediately** after forking, while the child process was programmed to sleep for 5 seconds. By the time the child woke up, its original parent was gone. I confirmed it had become an orphan by printing its new parent's PID (

os.getppid()), which had changed to 1 (the system's init process).

---

**Task 4: Inspecting Process Info from /proc**

I wrote a function

inspect_process that accepts a Process ID (PID) as an input. To get the required information, my script directly interacted with the

/proc virtual filesystem:

- I read and printed the

**Name, State, and VmSize** by opening and parsing lines from the /proc/[pid]/status file.

- I found the

**executable's full path** by using os.readlink() on the /proc/[pid]/exe symbolic link.

- I listed all

**open file descriptors** by using os.listdir() on the /proc/[pid]/fd directory.

---

**Task 5: Process Prioritization**

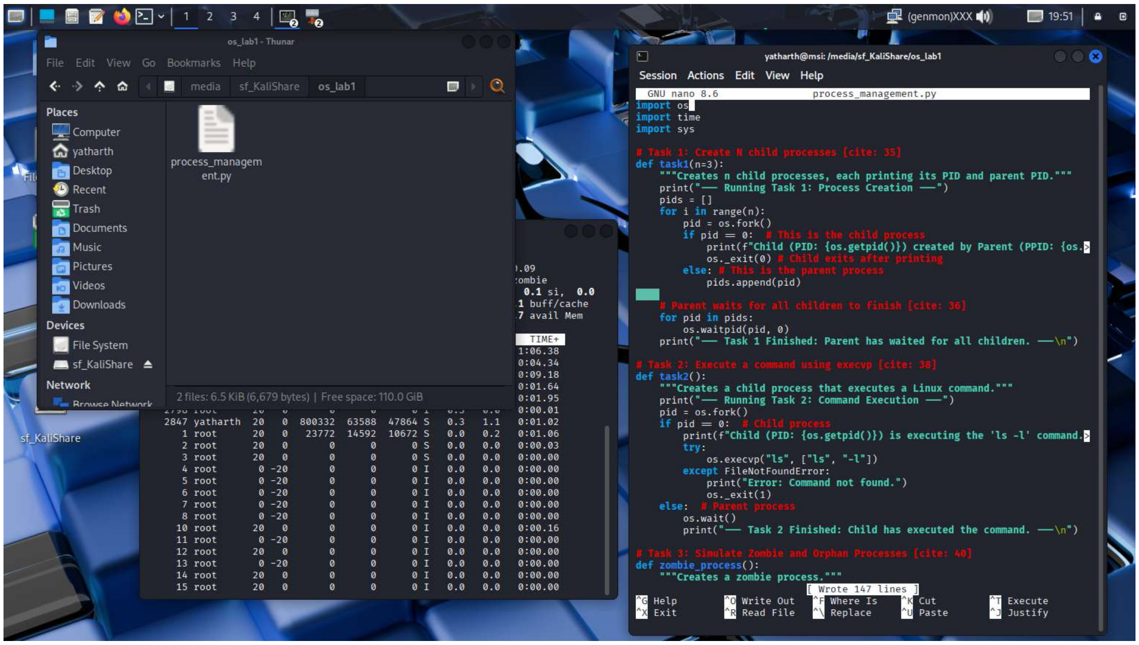
To demonstrate the effect of priority, my task5 function forked multiple child processes. Inside each child, I assigned a different priority using the

**os.nice()** call, with values of 0, 5, and 10. A lower

nice value corresponds to a higher priority. After setting the priority, each child performed an identical, CPU-intensive calculation (a large summation loop). By observing the output, I confirmed that the child with the highest priority (the lowest

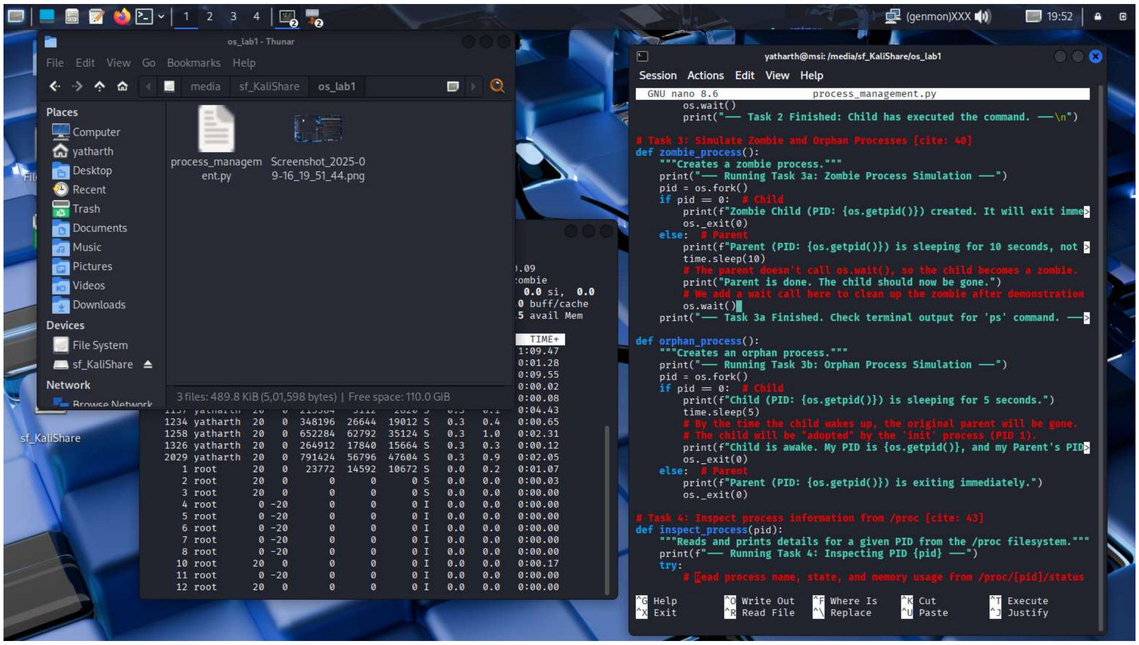
nice value) consistently finished its task first, showing the scheduler was giving it more CPU time

# Code snippets



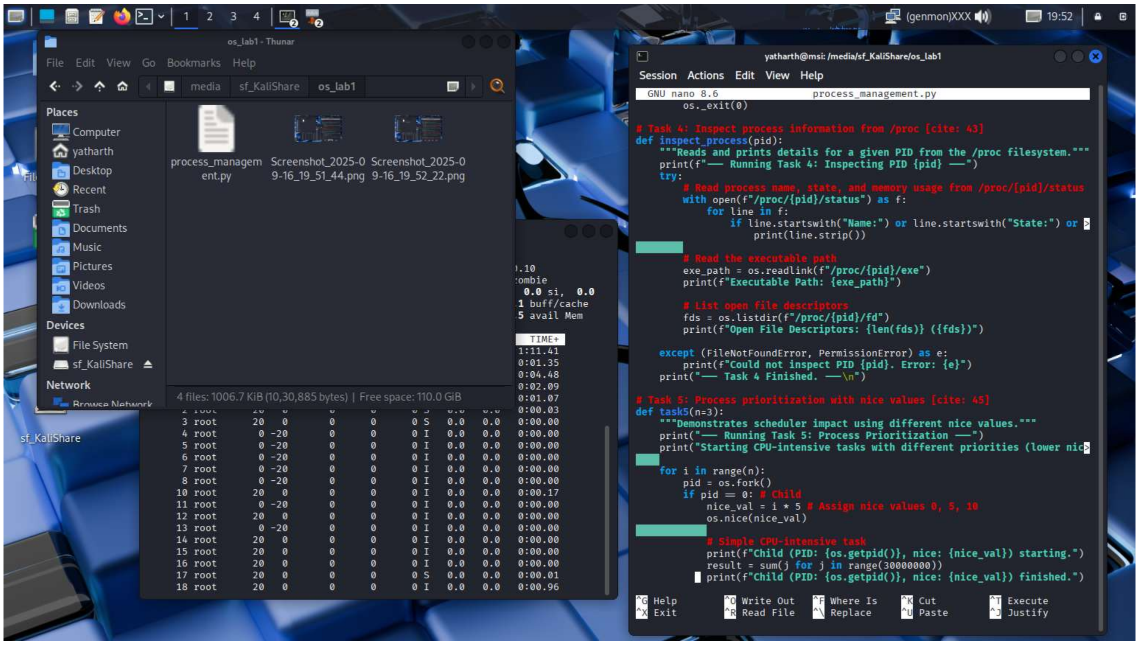
```
  GNU nano 8.6                    process_management.py
import os
import time
import sys

# Task 1: Create N child processes [cite: 35]
def task1(n=3):
    """Creates n child processes, each printing its PID and parent PID."""
    print("—— Running Task 1: Process Creation ——")
    pids = []
    for i in range(n):
        pid = os.fork()
        if pid == 0:  # This is the child process
            print(f"Child (PID: {os.getpid()}) created by Parent (PPID: {os.>
            os._exit(0) # Child exits after printing
        else: # This is the parent process
            pids.append(pid)

    # Parent waits for all children to finish [cite: 36]
    for pid in pids:
        os.waitpid(pid, 0)
    print("—— Task 1 Finished: Parent has waited for all children. ——\n")

# Task 2: Execute a command using execvp [cite: 38]
def task2():
    """Creates a child process that executes a Linux command."""
    print("—— Running Task 2: Command Execution ——")
    pid = os.fork()
    if pid == 0:  # Child process
        print(f"Child (PID: {os.getpid()}) is executing the 'ls -l' command.>
        try:
            os.execvp("ls", ["ls", "-l"])
        except FileNotFoundError:
            print("Error: Command not found.")
            os._exit(1)
    else:  # Parent process
        os.wait()
        print("—— Task 2 Finished: Child has executed the command. ——\n")

# Task 3: Simulate Zombie and Orphan Processes [cite: 40]
def zombie_process():
    """Creates a zombie process."""
                      [ Wrote 147 lines ]
^G Help       ^O Write Out   ^F Where Is    ^K Cut         ^T Execute
^X Exit       ^R Read File   ^\ Replace     ^U Paste       ^J Justify
```

```
.09
ombie
 0.1 si,  0.0
1 buff/cache
7 avail Mem

                                                                  TIME+
                                                                1:06.38
                                                                0:04.34
                                                                0:09.18
                                                                0:01.64
                                                                0:01.95
                                                                0:00.01
2847 yatharth  20   0  800332  63588  47864 S   0.3   1.1   0:01.02
   1 root      20   0   23772  14592  10672 S   0.0   0.2   0:01.06
   2 root      20   0       0      0      0 S   0.0   0.0   0:00.03
   3 root      20   0       0      0      0 S   0.0   0.0   0:00.00
   4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
   5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
   6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
   7 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
   8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
  10 root      20   0       0      0      0 I   0.0   0.0   0:00.16
  11 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
  12 root      20   0       0      0      0 I   0.0   0.0   0:00.00
  13 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
  14 root      20   0       0      0      0 I   0.0   0.0   0:00.00
  15 root      20   0       0      0      0 I   0.0   0.0   0:00.00
```


os_lab1 - Thunar
File Edit View Go Bookmarks Help
media sf_KaliShare os_lab1
Places
Computer
yatharth
Desktop
Recent
Trash
Documents
Music
Pictures
Videos
Downloads
Devices
File System
sf_KaliShare
Network
Browse Network
process_management.py
Screenshot_2025-09-16_19_51_44.png
3 files: 489.8 KiB (5,01,598 bytes) | Free space: 110.0 GiB
sf_KaliShare
(genmon)XXX
19:52
yatharth@msi: /media/sf_KaliShare/os_lab1
Session Actions Edit View Help

```
  GNU nano 8.6                    process_management.py
        os.wait()
        print("—— Task 2 Finished: Child has executed the command. ——\n")

# Task 3: Simulate Zombie and Orphan Processes [cite: 40]
def zombie_process():
    """Creates a zombie process."""
    print("—— Running Task 3a: Zombie Process Simulation ——")
    pid = os.fork()
    if pid == 0:  # Child
        print(f"Zombie Child (PID: {os.getpid()}) created. It will exit imme>
        os._exit(0)
    else:  # Parent
        print(f"Parent (PID: {os.getpid()}) is sleeping for 10 seconds, not >
        time.sleep(10)
        # The parent doesn't call os.wait(), so the child becomes a zombie.
        print("Parent is done. The child should now be gone.")
        # We add a wait call here to clean up the zombie after demonstration
        os.wait()
    print("—— Task 3a Finished. Check terminal output for 'ps' command. ——>

def orphan_process():
    """Creates an orphan process."""
    print("—— Running Task 3b: Orphan Process Simulation ——")
    pid = os.fork()
    if pid == 0:  # Child
        print(f"Child (PID: {os.getpid()}) is sleeping for 5 seconds.")
        time.sleep(5)
        # By the time the child wakes up, the original parent will be gone.
        # The child will be "adopted" by the 'init' process (PID 1).
        print(f"Child is awake. My PID is {os.getpid()}, and my Parent's PID>
        os._exit(0)
    else:  # Parent
        print(f"Parent (PID: {os.getpid()}) is exiting immediately.")
        os._exit(0)

# Task 4: Inspect process information from /proc [cite: 43]
def inspect_process(pid):
    """Reads and prints details for a given PID from the /proc filesystem."""
    print(f"—— Running Task 4: Inspecting PID {pid} ——")
    try:
        # Read process name, state, and memory usage from /proc/[pid]/status

^G Help       ^O Write Out   ^F Where Is    ^K Cut         ^T Execute
^X Exit       ^R Read File   ^\ Replace     ^U Paste       ^J Justify
```

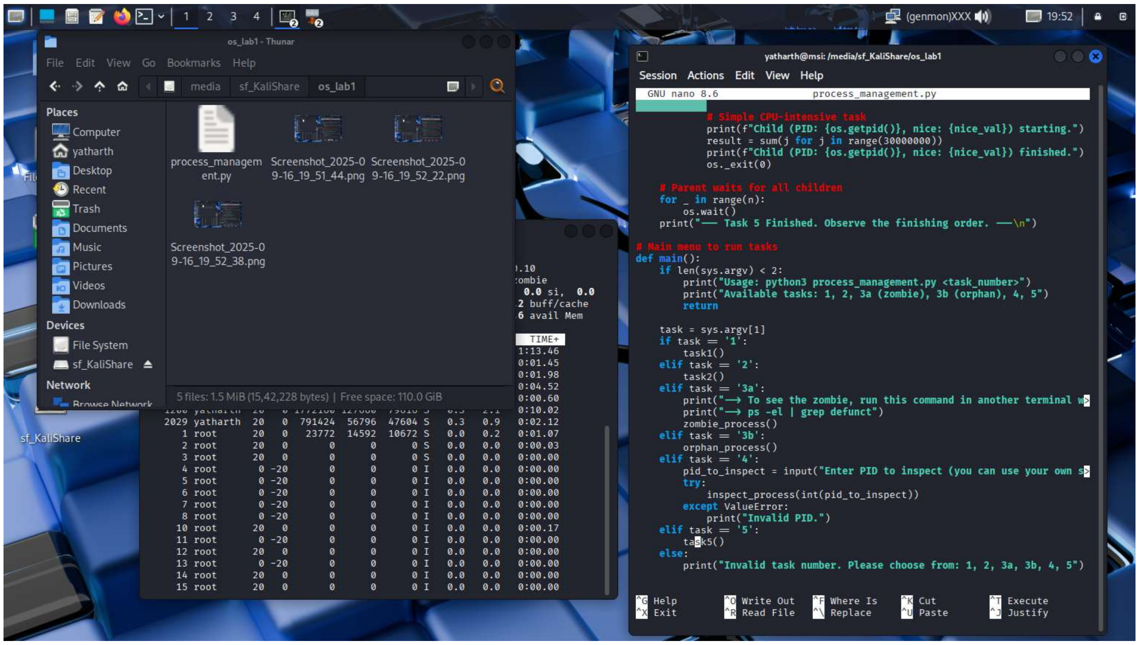
```
.09
ombie
 0.0 si,  0.0
0 buff/cache
5 avail Mem

                                                                  TIME+
                                                                1:09.47
                                                                0:01.28
                                                                0:09.55
                                                                0:00.02
                                                                0:00.08
                                                                0:04.43
1234 yatharth  20   0  348196  26644  19012 S   0.3   0.4   0:00.65
1258 yatharth  20   0  652284  62792  35124 S   0.3   1.0   0:02.31
1326 yatharth  20   0  264912  17840  15664 S   0.3   0.3   0:00.12
2029 yatharth  20   0  791424  56796  47604 S   0.3   0.9   0:02.05
   1 root      20   0   23772  14592  10672 S   0.0   0.2   0:01.07
   2 root      20   0       0      0      0 S   0.0   0.0   0:00.03
   3 root      20   0       0      0      0 S   0.0   0.0   0:00.00
   4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
   5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
   6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
   7 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
   8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
  10 root      20   0       0      0      0 I   0.0   0.0   0:00.17
  11 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
  12 root      20   0       0      0      0 I   0.0   0.0   0:00.00
```



4 files: 1006.7 KiB (10,30,885 bytes) | Free space: 110.0 GiB

```
).10
zombie
 0.0 si,  0.0
1 buff/cache
5 avail Mem

 TIME+
1:11.41
0:01.35
0:04.48
0:02.09
0:01.07
0:00.03
  3 root      20   0       0      0      0 S   0.0   0.0   0:00.00
  4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
  5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
  6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
  7 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
  8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
 10 root      20   0       0      0      0 I   0.0   0.0   0:00.17
 11 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
 12 root      20   0       0      0      0 I   0.0   0.0   0:00.00
 13 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
 14 root      20   0       0      0      0 I   0.0   0.0   0:00.00
 15 root      20   0       0      0      0 I   0.0   0.0   0:00.00
 16 root      20   0       0      0      0 I   0.0   0.0   0:00.00
 17 root      20   0       0      0      0 S   0.0   0.0   0:00.01
 18 root      20   0       0      0      0 I   0.0   0.0   0:00.96
```

```
  GNU nano 8.6                    process_management.py
        os._exit(0)

# Task 4: Inspect process information from /proc [cite: 43]
def inspect_process(pid):
    """Reads and prints details for a given PID from the /proc filesystem."""
    print(f"—— Running Task 4: Inspecting PID {pid} ——")
    try:
        # Read process name, state, and memory usage from /proc/[pid]/status
        with open(f"/proc/{pid}/status") as f:
            for line in f:
                if line.startswith("Name:") or line.startswith("State:") or >
                    print(line.strip())

        # Read the executable path
        exe_path = os.readlink(f"/proc/{pid}/exe")
        print(f"Executable Path: {exe_path}")

        # List open file descriptors
        fds = os.listdir(f"/proc/{pid}/fd")
        print(f"Open File Descriptors: {len(fds)} ({fds})")

    except (FileNotFoundError, PermissionError) as e:
        print(f"Could not inspect PID {pid}. Error: {e}")
    print("—— Task 4 Finished. ——\n")

# Task 5: Process prioritization with nice values [cite: 45]
def task5(n=3):
    """Demonstrates scheduler impact using different nice values."""
    print("—— Running Task 5: Process Prioritization ——")
    print("Starting CPU-intensive tasks with different priorities (lower nic>

    for i in range(n):
        pid = os.fork()
        if pid == 0: # Child
            nice_val = i * 5 # Assign nice values 0, 5, 10
            os.nice(nice_val)

            # Simple CPU-intensive task
            print(f"Child (PID: {os.getpid()}, nice: {nice_val}) starting.")
            result = sum(j for j in range(30000000))
            print(f"Child (PID: {os.getpid()}, nice: {nice_val}) finished.")

^G Help         ^O Write Out    ^F Where Is     ^K Cut          ^T Execute
^X Exit         ^R Read File    ^\ Replace      ^U Paste        ^J Justify
```


1 2 3 4
(genmon)XXX
19:52
os_lab1 - Thunar
File Edit View Go Bookmarks Help
media sf_KaliShare os_lab1
Places
Computer
yatharth
Desktop
Recent
Trash
Documents
Music
Pictures
Videos
Downloads
Devices
File System
sf_KaliShare
Network
Browse Network
process_management.py
Screenshot_2025-09-16_19_51_44.png
Screenshot_2025-09-16_19_52_22.png
Screenshot_2025-09-16_19_52_38.png
sf_KaliShare
yatharth@msi: /media/sf_KaliShare/os_lab1
Session Actions Edit View Help

5 files: 1.5 MiB (15,42,228 bytes) | Free space: 110.0 GiB

```
).10
zombie
 0.0 si,  0.0
2 buff/cache
6 avail Mem

 TIME+
1:13.46
0:01.45
0:01.98
0:04.52
0:00.60
0:10.02
2029 yatharth  20   0  791424  56796  47604 S   0.3   0.9   0:02.12
   1 root      20   0   23772  14592  10672 S   0.0   0.2   0:01.07
   2 root      20   0       0      0      0 S   0.0   0.0   0:00.03
   3 root      20   0       0      0      0 S   0.0   0.0   0:00.00
   4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
   5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
   6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
   7 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
   8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
  10 root      20   0       0      0      0 I   0.0   0.0   0:00.17
  11 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
  12 root      20   0       0      0      0 I   0.0   0.0   0:00.00
  13 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
  14 root      20   0       0      0      0 I   0.0   0.0   0:00.00
  15 root      20   0       0      0      0 I   0.0   0.0   0:00.00
```

```
  GNU nano 8.6                    process_management.py

            # Simple CPU-intensive task
            print(f"Child (PID: {os.getpid()}, nice: {nice_val}) starting.")
            result = sum(j for j in range(30000000))
            print(f"Child (PID: {os.getpid()}, nice: {nice_val}) finished.")
            os._exit(0)

    # Parent waits for all children
    for _ in range(n):
        os.wait()
    print("—— Task 5 Finished. Observe the finishing order. ——\n")

# Main menu to run tasks
def main():
    if len(sys.argv) < 2:
        print("Usage: python3 process_management.py <task_number>")
        print("Available tasks: 1, 2, 3a (zombie), 3b (orphan), 4, 5")
        return

    task = sys.argv[1]
    if task == '1':
        task1()
    elif task == '2':
        task2()
    elif task == '3a':
        print("⟶ To see the zombie, run this command in another terminal w>
        print("⟶ ps -el | grep defunct")
        zombie_process()
    elif task == '3b':
        orphan_process()
    elif task == '4':
        pid_to_inspect = input("Enter PID to inspect (you can use your own s>
        try:
            inspect_process(int(pid_to_inspect))
        except ValueError:
            print("Invalid PID.")
    elif task == '5':
        task5()
    else:
        print("Invalid task number. Please choose from: 1, 2, 3a, 3b, 4, 5")

^G Help         ^O Write Out    ^F Where Is     ^K Cut          ^T Execute
^X Exit         ^R Read File    ^\ Replace      ^U Paste        ^J Justify
```