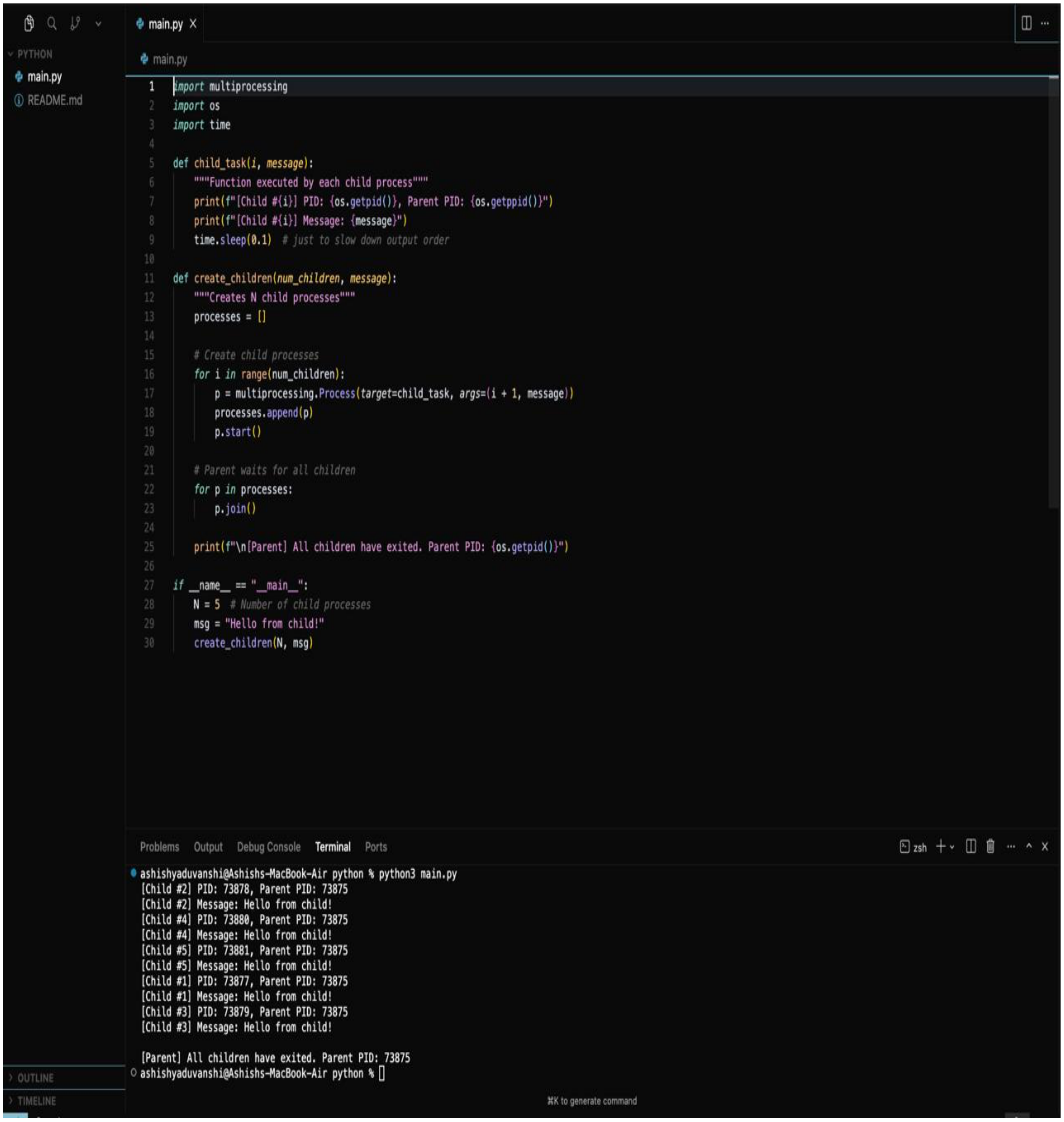
1. Write a Python program that creates N child processes using os.fork(). Each child prints:
   - Its PID
   - Its Parent PID
   - A custom message

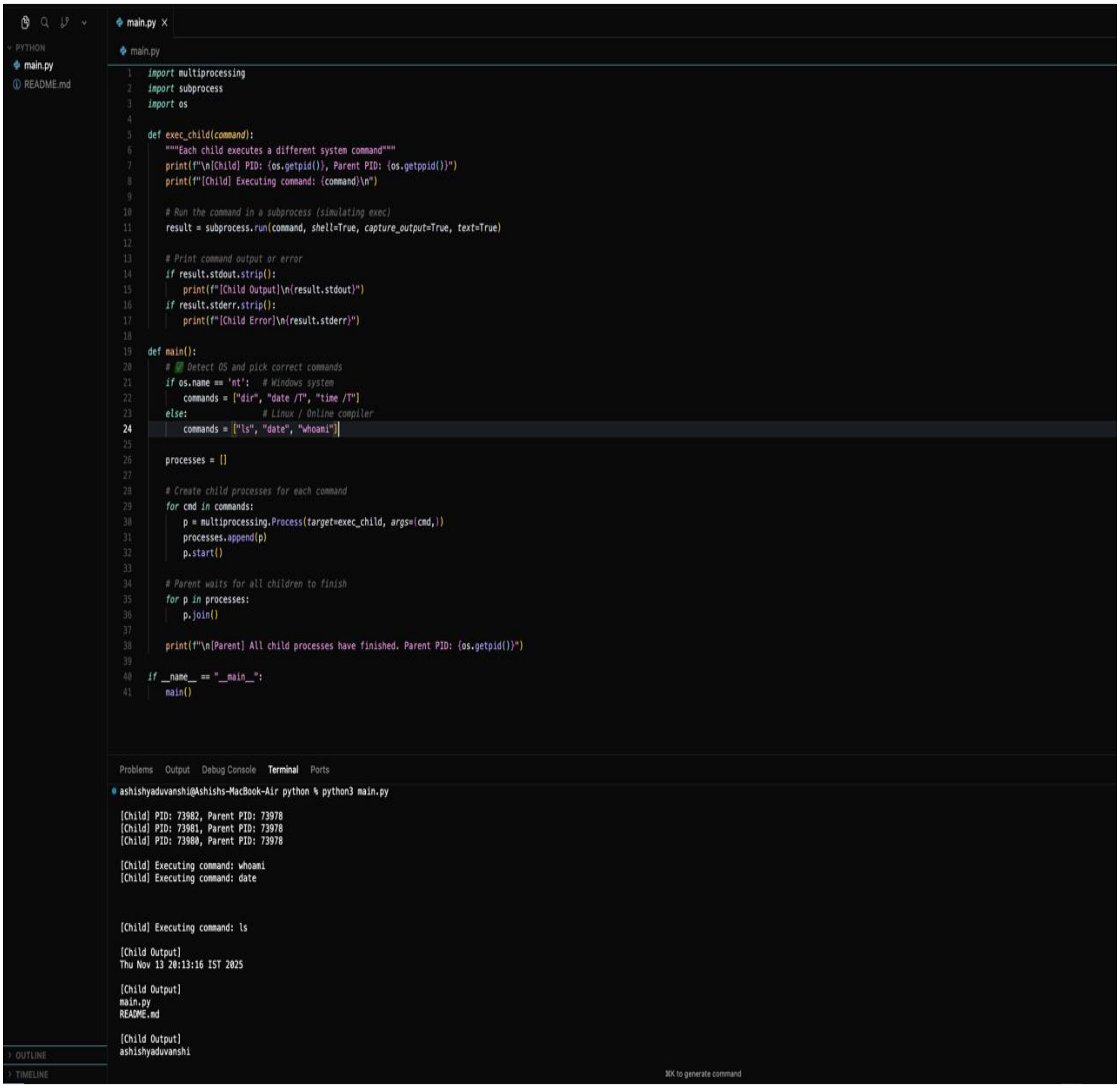
   The parent should wait for all children using os.wait().



```python
import multiprocessing
import os
import time

def child_task(i, message):
    """Function executed by each child process"""
    print(f"[Child #{i}] PID: {os.getpid()}, Parent PID: {os.getppid()}")
    print(f"[Child #{i}] Message: {message}")
    time.sleep(0.1)  # just to slow down output order

def create_children(num_children, message):
    """Creates N child processes"""
    processes = []

    # Create child processes
    for i in range(num_children):
        p = multiprocessing.Process(target=child_task, args=(i + 1, message))
        processes.append(p)
        p.start()

    # Parent waits for all children
    for p in processes:
        p.join()

    print(f"\n[Parent] All children have exited. Parent PID: {os.getpid()}")

if __name__ == "__main__":
    N = 5  # Number of child processes
    msg = "Hello from child!"
    create_children(N, msg)
```

```
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py
[Child #2] PID: 73878, Parent PID: 73875
[Child #2] Message: Hello from child!
[Child #4] PID: 73880, Parent PID: 73875
[Child #4] Message: Hello from child!
[Child #5] PID: 73881, Parent PID: 73875
[Child #5] Message: Hello from child!
[Child #1] PID: 73877, Parent PID: 73875
[Child #1] Message: Hello from child!
[Child #3] PID: 73879, Parent PID: 73875
[Child #3] Message: Hello from child!

[Parent] All children have exited. Parent PID: 73875
ashishyaduvanshi@Ashishs-MacBook-Air python % 
```

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().

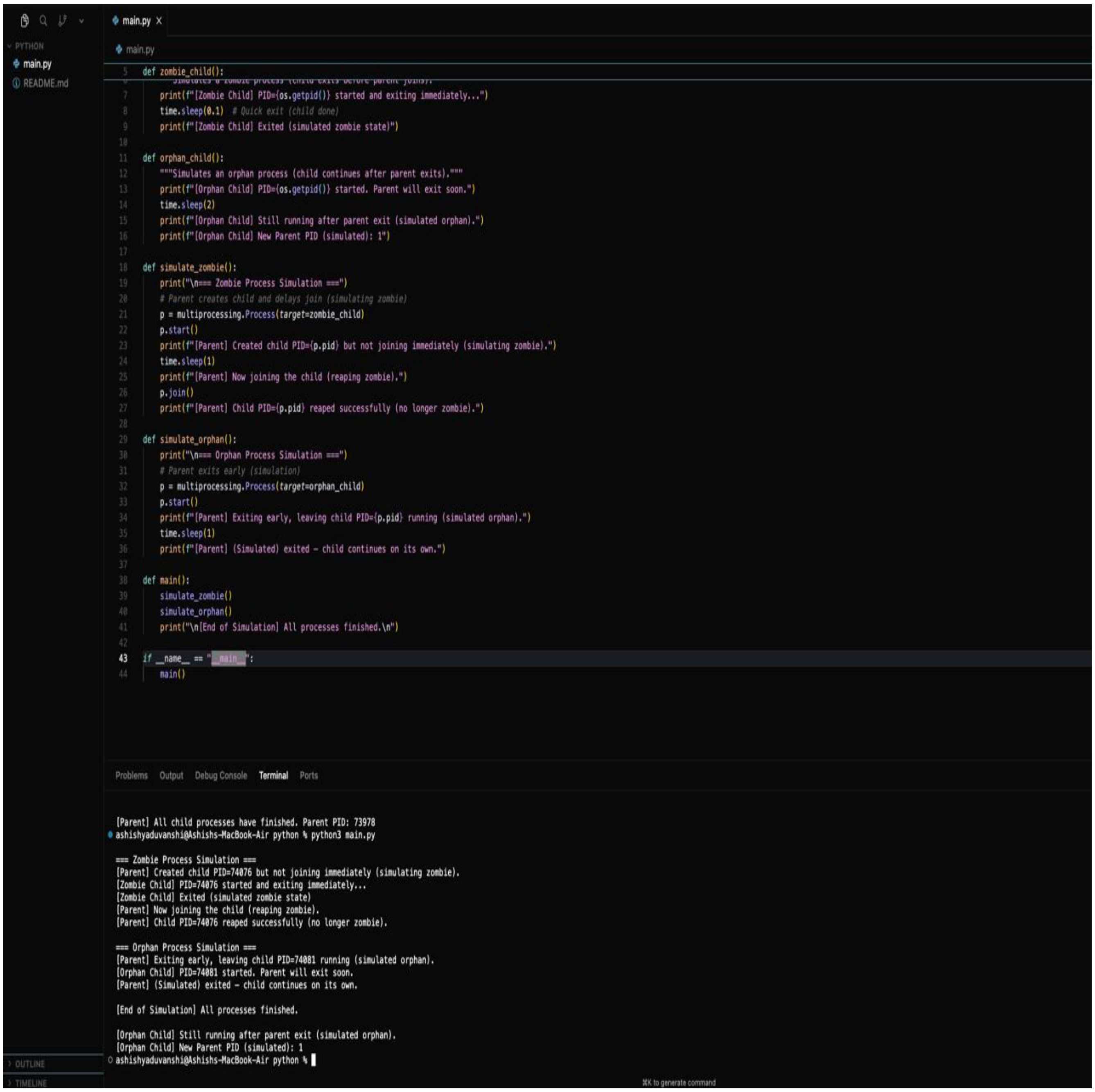


```python
import multiprocessing
import subprocess
import os

def exec_child(command):
    """Each child executes a different system command"""
    print(f"\n[Child] PID: {os.getpid()}, Parent PID: {os.getppid()}")
    print(f"[Child] Executing command: {command}\n")

    # Run the command in a subprocess (simulating exec)
    result = subprocess.run(command, shell=True, capture_output=True, text=True)

    # Print command output or error
    if result.stdout.strip():
        print(f"[Child Output]\n{result.stdout}")
    if result.stderr.strip():
        print(f"[Child Error]\n{result.stderr}")

def main():
    # ✅ Detect OS and pick correct commands
    if os.name == 'nt':   # Windows system
        commands = ["dir", "date /T", "time /T"]
    else:                 # Linux / Online compiler
        commands = ["ls", "date", "whoami"]

    processes = []

    # Create child processes for each command
    for cmd in commands:
        p = multiprocessing.Process(target=exec_child, args=(cmd,))
        processes.append(p)
        p.start()

    # Parent waits for all children to finish
    for p in processes:
        p.join()

    print(f"\n[Parent] All child processes have finished. Parent PID: {os.getpid()}")

if __name__ == "__main__":
    main()
```

```
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py

[Child] PID: 73982, Parent PID: 73978
[Child] PID: 73981, Parent PID: 73978
[Child] PID: 73980, Parent PID: 73978

[Child] Executing command: whoami
[Child] Executing command: date



[Child] Executing command: ls

[Child Output]
Thu Nov 13 20:13:16 IST 2025

[Child Output]
main.py
README.md

[Child Output]
ashishyaduvanshi
```

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.

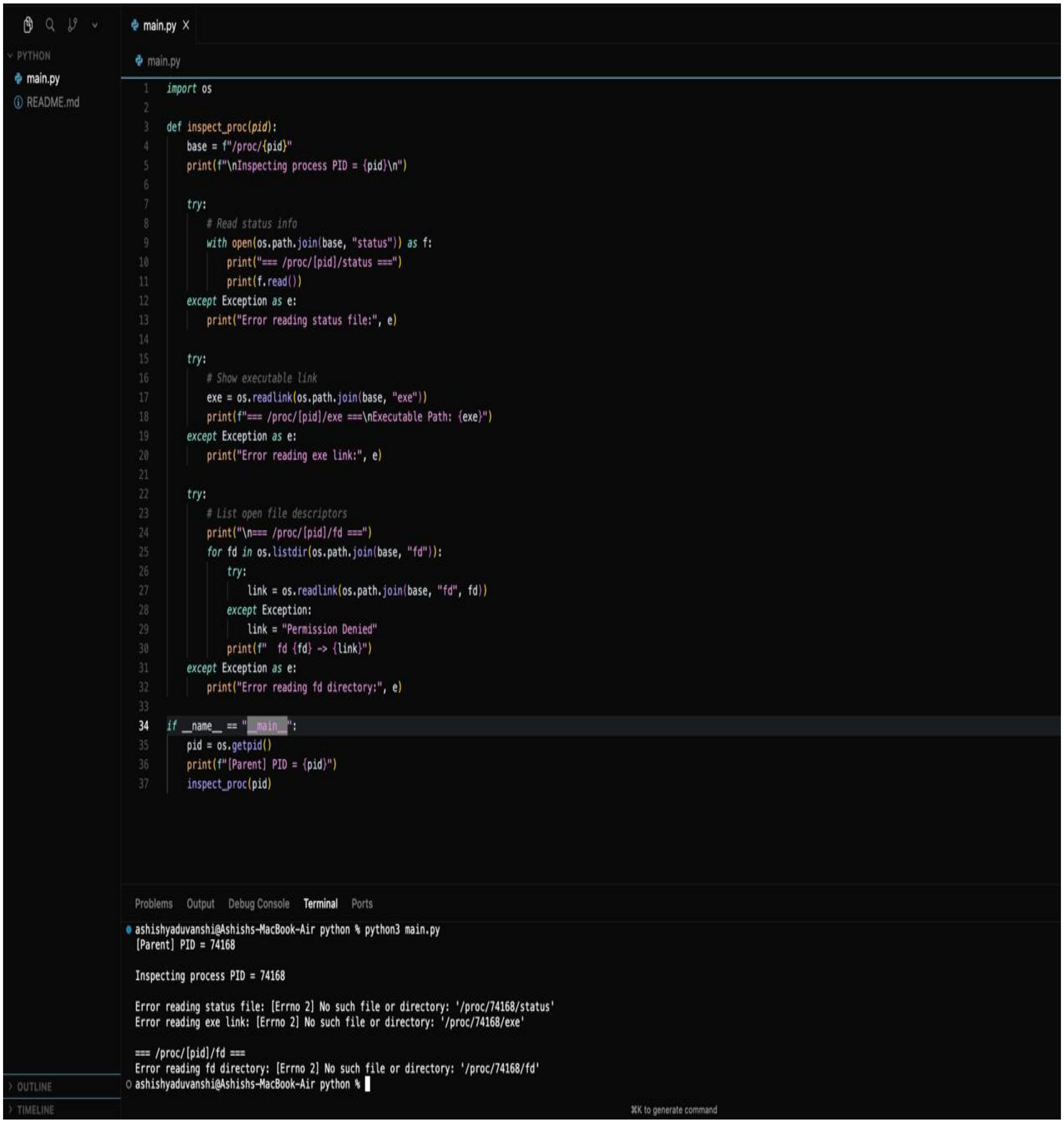


```python
def zombie_child():
    """Simulates a zombie process (child exits before parent joins)."""
    print(f"[Zombie Child] PID={os.getpid()} started and exiting immediately...")
    time.sleep(0.1)  # Quick exit (child done)
    print(f"[Zombie Child] Exited (simulated zombie state)")

def orphan_child():
    """Simulates an orphan process (child continues after parent exits)."""
    print(f"[Orphan Child] PID={os.getpid()} started. Parent will exit soon.")
    time.sleep(2)
    print(f"[Orphan Child] Still running after parent exit (simulated orphan).")
    print(f"[Orphan Child] New Parent PID (simulated): 1")

def simulate_zombie():
    print("\n=== Zombie Process Simulation ===")
    # Parent creates child and delays join (simulating zombie)
    p = multiprocessing.Process(target=zombie_child)
    p.start()
    print(f"[Parent] Created child PID={p.pid} but not joining immediately (simulating zombie).")
    time.sleep(1)
    print(f"[Parent] Now joining the child (reaping zombie).")
    p.join()
    print(f"[Parent] Child PID={p.pid} reaped successfully (no longer zombie).")

def simulate_orphan():
    print("\n=== Orphan Process Simulation ===")
    # Parent exits early (simulation)
    p = multiprocessing.Process(target=orphan_child)
    p.start()
    print(f"[Parent] Exiting early, leaving child PID={p.pid} running (simulated orphan).")
    time.sleep(1)
    print(f"[Parent] (Simulated) exited – child continues on its own.")

def main():
    simulate_zombie()
    simulate_orphan()
    print("\n[End of Simulation] All processes finished.\n")

if __name__ == "__main__":
    main()
```

```
[Parent] All child processes have finished. Parent PID: 73978
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py

=== Zombie Process Simulation ===
[Parent] Created child PID=74076 but not joining immediately (simulating zombie).
[Zombie Child] PID=74076 started and exiting immediately...
[Zombie Child] Exited (simulated zombie state)
[Parent] Now joining the child (reaping zombie).
[Parent] Child PID=74076 reaped successfully (no longer zombie).

=== Orphan Process Simulation ===
[Parent] Exiting early, leaving child PID=74081 running (simulated orphan).
[Orphan Child] PID=74081 started. Parent will exit soon.
[Parent] (Simulated) exited – child continues on its own.

[End of Simulation] All processes finished.

[Orphan Child] Still running after parent exit (simulated orphan).
[Orphan Child] New Parent PID (simulated): 1
ashishyaduvanshi@Ashishs-MacBook-Air python %
```

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

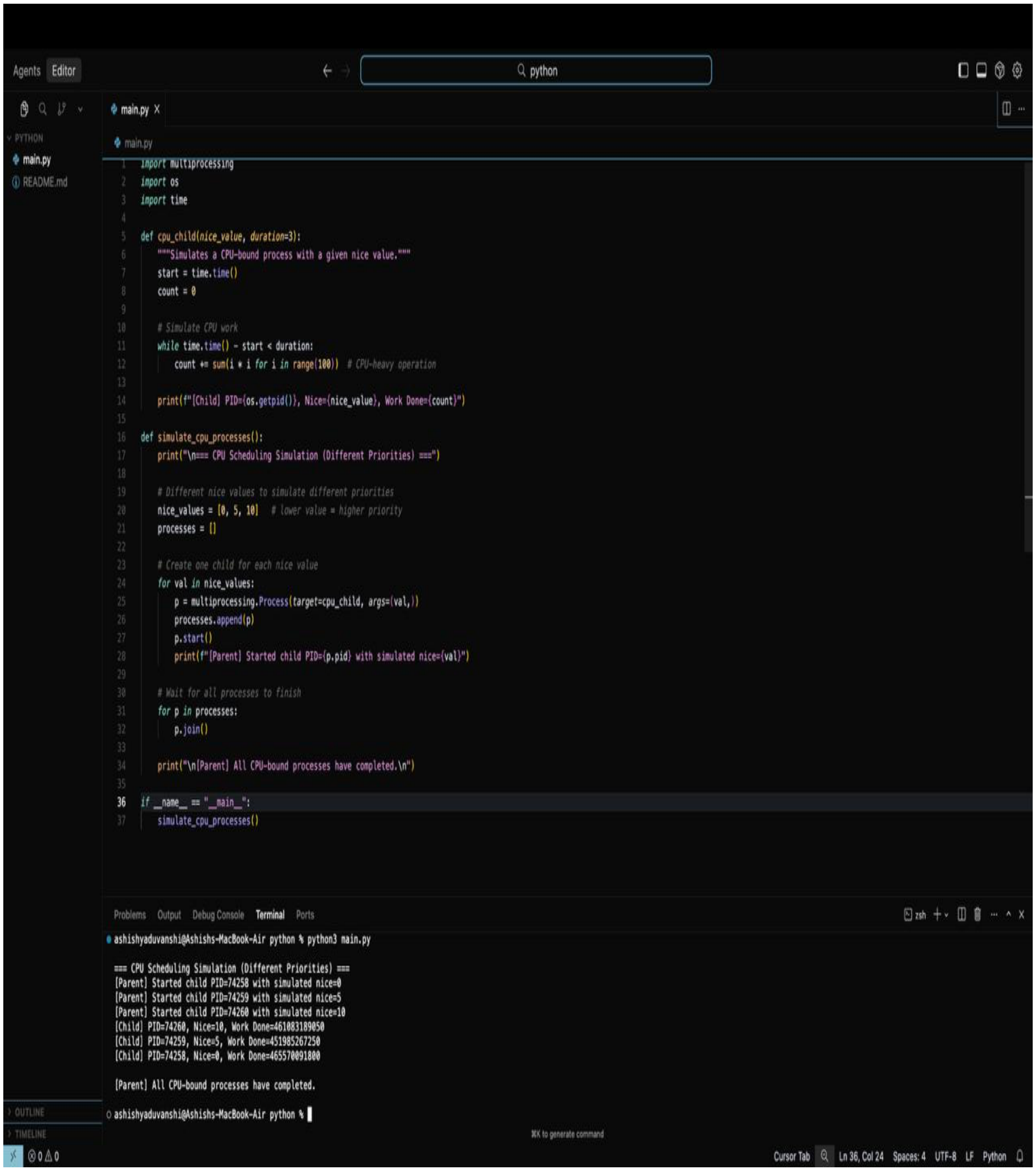


```python
import os

def inspect_proc(pid):
    base = f"/proc/{pid}"
    print(f"\nInspecting process PID = {pid}\n")

    try:
        # Read status info
        with open(os.path.join(base, "status")) as f:
            print("=== /proc/[pid]/status ===")
            print(f.read())
    except Exception as e:
        print("Error reading status file:", e)

    try:
        # Show executable link
        exe = os.readlink(os.path.join(base, "exe"))
        print(f"=== /proc/[pid]/exe ===\nExecutable Path: {exe}")
    except Exception as e:
        print("Error reading exe link:", e)

    try:
        # List open file descriptors
        print("\n=== /proc/[pid]/fd ===")
        for fd in os.listdir(os.path.join(base, "fd")):
            try:
                link = os.readlink(os.path.join(base, "fd", fd))
            except Exception:
                link = "Permission Denied"
            print(f"  fd {fd} -> {link}")
    except Exception as e:
        print("Error reading fd directory:", e)

if __name__ == "__main__":
    pid = os.getpid()
    print(f"[Parent] PID = {pid}")
    inspect_proc(pid)
```

```
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py
[Parent] PID = 74168

Inspecting process PID = 74168

Error reading status file: [Errno 2] No such file or directory: '/proc/74168/status'
Error reading exe link: [Errno 2] No such file or directory: '/proc/74168/exe'

=== /proc/[pid]/fd ===
Error reading fd directory: [Errno 2] No such file or directory: '/proc/74168/fd'
ashishyaduvanshi@Ashishs-MacBook-Air python %
```

Task 5: Process Prioritization

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



```python
import multiprocessing
import os
import time

def cpu_child(nice_value, duration=3):
    """Simulates a CPU-bound process with a given nice value."""
    start = time.time()
    count = 0

    # Simulate CPU work
    while time.time() - start < duration:
        count += sum(i * i for i in range(100))  # CPU-heavy operation

    print(f"[Child] PID={os.getpid()}, Nice={nice_value}, Work Done={count}")

def simulate_cpu_processes():
    print("\n=== CPU Scheduling Simulation (Different Priorities) ===")

    # Different nice values to simulate different priorities
    nice_values = [0, 5, 10]   # lower value = higher priority
    processes = []

    # Create one child for each nice value
    for val in nice_values:
        p = multiprocessing.Process(target=cpu_child, args=(val,))
        processes.append(p)
        p.start()
        print(f"[Parent] Started child PID={p.pid} with simulated nice={val}")

    # Wait for all processes to finish
    for p in processes:
        p.join()

    print("\n[Parent] All CPU-bound processes have completed.\n")

if __name__ == "__main__":
    simulate_cpu_processes()
```

```
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py

=== CPU Scheduling Simulation (Different Priorities) ===
[Parent] Started child PID=74258 with simulated nice=0
[Parent] Started child PID=74259 with simulated nice=5
[Parent] Started child PID=74260 with simulated nice=10
[Child] PID=74260, Nice=10, Work Done=461083189050
[Child] PID=74259, Nice=5, Work Done=451985267250
[Child] PID=74258, Nice=0, Work Done=465570091800

[Parent] All CPU-bound processes have completed.

ashishyaduvanshi@Ashishs-MacBook-Air python %
```