

KR MANGALAM UNIVERSITY

SOHNA ROAD, GURUGRAM



OPERATING SYSTEM LAB FILE

COURSE CODE-ENCS351

Submitted by

Name: Aldrin Debbarma
No.: 2401011482

Submitted To

Dr Satinder Pal Singh Roll

Program: BTech CSE
Section: 6/F
:Session: 2025/26
Date: 02/12/2025

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()`.

The screenshot shows a code editor with a Python file named `main.py` open. The code uses the `multiprocessing` module to create multiple child processes. Each child prints its own PID, its parent's PID, and a custom message. The parent process waits for all children to exit before printing a final message. The terminal below shows the execution of the script and its output.

```

1 import multiprocessing
2 import os
3 import time
4
5 def child_task(i, message):
6     """Function executed by each child process"""
7     print(f"[Child #{i}] PID: {os.getpid()}, Parent PID: {os.getppid()}")
8     print(f"[Child #{i}] Message: {message}")
9     time.sleep(0.1) # just to slow down output order
10
11 def create_children(num_children, message):
12     """Creates N child processes"""
13     processes = []
14
15     # Create child processes
16     for i in range(num_children):
17         p = multiprocessing.Process(target=child_task, args=(i + 1, message))
18         processes.append(p)
19         p.start()
20
21     # Parent waits for all children
22     for p in processes:
23         p.join()
24
25     print("\n[Parent] All children have exited. Parent PID: {os.getpid()}")
26
27 if __name__ == "__main__":
28     N = 5 # Number of child processes
29     msg = "Hello from child!"
30     create_children(N, msg)

```

Terminal Output:

```

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().

```
main.py X
PYTHON
main.py
README.md

1 import multiprocessing
2 import subprocess
3 import os
4
5 def exec_child(command):
6     """Each child executes a different system command"""
7     print(f"\n[Child] PID: {os.getpid()}, Parent PID: {os.getppid()}")
8     print(f"[Child] Executing command: {command}\n")
9
10    # Run the command in a subprocess (simulating exec)
11    result = subprocess.run(command, shell=True, capture_output=True, text=True)
12
13    # Print command output or error
14    if result.stdout.strip():
15        print(f"[Child Output]\n{result.stdout}")
16    if result.stderr.strip():
17        print(f"[Child Error]\n{result.stderr}")
18
19 def main():
20     # Detect OS and pick correct commands
21     if os.name == 'nt': # Windows system
22         commands = ["dir", "date /T", "time /T"]
23     else: # Linux / Online compiler
24         commands = ["ls", "date", "whoami"]
25
26     processes = []
27
28     # Create child processes for each command
29     for cmd in commands:
30         p = multiprocessing.Process(target=exec_child, args=(cmd,))
31         processes.append(p)
32         p.start()
33
34     # Parent waits for all children to finish
35     for p in processes:
36         p.join()
37
38     print("\n[Parent] All child processes have finished. Parent PID: {os.getpid()}")
39
40 if __name__ == "__main__":
41     main()

Problems Output Debug Console Terminal Ports
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
WK to generate command
OUTLINE
TIMELINE
```

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.

The screenshot shows a code editor with a Python file named `main.py`. The code defines several functions: `zombie_child()`, `orphan_child()`, `simulate_zombie()`, `simulate_orphan()`, and `main()`. The `zombie_child()` function creates a child process, prints its PID, sleeps for 0.1 seconds, and then prints it has exited. The `orphan_child()` function creates a child process, prints its PID, sleeps for 2 seconds, and then prints it is still running. The `simulate_zombie()` function creates a child process, sleeps for 1 second, joins it, and reaps it. The `simulate_orphan()` function creates a child process, sleeps for 1 second, and then exits. The `main()` function calls both simulation functions and prints a message when they finish.

```
def zombie_child():
    print("[Zombie Child] PID={os.getpid()} started and exiting immediately...")
    time.sleep(0.1) # Quick exit (child done)
    print("[Zombie Child] Exited (simulated zombie state)")

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

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

def simulate_orphan():
    print("== Orphan Process Simulation ==")
    # Parent exits early (simulation)
    p = multiprocessing.Process(target=orphan_child)
    p.start()
    print("[Parent] Exiting early, leaving child PID={p.pid} running (simulated orphan).")
    time.sleep(1)
    print("[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()
```

The terminal output shows the execution of `main.py` and the resulting simulated process behavior:

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

The screenshot shows a Python code editor with a file named `main.py`. The code defines a function `inspect_proc(pid)` that performs three main tasks: reading process status info, reading the executable link, and listing open file descriptors. It handles errors for each task and prints the results. The code includes imports for `os`, defines the function, and includes a check for the main module to run it with `os.getpid()`.

```
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)
```

Below the code editor is a terminal window showing the execution of the script and its output. The terminal shows the user's command `ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py`, the process ID `[Parent] PID = 74168`, and the inspection results for PID 74168. The results indicate that the status and executable files were not found, while the fd directory was successfully listed.

```
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'
```

Task 5: Process Prioritization

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

```
Agents Editor ← → Q python
 PYTHON
 main.py README.md
 main.py
 1 import multiprocessing
 2 import os
 3 import time
 4
 5 def cpu_child(nice_value, duration=3):
 6     """Simulates a CPU-bound process with a given nice value."""
 7     start = time.time()
 8     count = 0
 9
10     # Simulate CPU work
11     while time.time() - start < duration:
12         count += sum(i * i for i in range(100)) # CPU-heavy operation
13
14     print(f"[Child] PID={os.getpid()}, Nice={nice_value}, Work Done={count}")
15
16 def simulate_cpu_processes():
17     print("\n*** CPU Scheduling Simulation (Different Priorities) ***")
18
19     # Different nice values to simulate different priorities
20     nice_values = [0, 5, 10] # lower value = higher priority
21     processes = []
22
23     # Create one child for each nice value
24     for val in nice_values:
25         p = multiprocessing.Process(target=cpu_child, args=(val,))
26         processes.append(p)
27         p.start()
28         print(f"[Parent] Started child PID={p.pid} with simulated nice={val}")
29
30     # Wait for all processes to finish
31     for p in processes:
32         p.join()
33
34     print("\n[Parent] All CPU-bound processes have completed.\n")
35
36 if __name__ == "__main__":
37     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=461083189850
[Child] PID=74259, Nice=5, Work Done=451985267250
[Child] PID=74258, Nice=0, Work Done=465570091880

[Parent] All CPU-bound processes have completed.
```

Cursor Tab ↵ Ln 36, Col 24 Spaces: 4 UTF-8 LF Python ⌂