ASSIGNMENT – 1

OPERATING SYSTEM LAB – WORK

Name: Aryan Shukla

Roll No. : 2301410018

Course: BTech CSE (cyber security)

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().

CODE:

OUTPUT

Task 2:

```
└─$ python3 task1_process_creation.py

Enter the number of child processes to create: 3
Child 1: PID=12408, Parent PID=12367
Child 1: Hello from child process!
Child 2: PID=12409, Parent PID=12367
Child 2: Hello from child process!
Child 3: PID=12410, Parent PID=12367
Child 3: Hello from child process!
Parent: Child with PID 12408 finished with status 0
Parent: Child with PID 12409 finished with status 0
Parent: Child with PID 12410 finished with status 0
```

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().

CODE:

```
GNU nano 8.4                                    task2_command_exec.py
import os

def main():
    commands = ["ls", "date", "whoami"]  # Commands children will run

    N = len(commands)
    for i in range(N):
        pid = os.fork()
        if pid == 0:
            print(f"Child {i+1}: PID={os.getpid()}, executing '{commands[i]}'")
            os.execvp(commands[i], [commands[i]])  # Replace child process
        # Parent continues loop

    # Parent waits for all children
    for _ in range(N):
        os.wait()

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

OUTPUT

```
└─$ python3 task2_command_exec.py
Child 1: PID=14587, executing 'ls'
Child 2: PID=14588, executing 'date'
Monday 08 September 2025 03:08:32 PM IST
Child 3: PID=14589, executing 'whoami'
task1_process_creation.py  task2_command_exec.py  task3_zombie_orphan.py  task4_proc_inspection.py  task5_priority.py
```

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.

CODE

Task 3:

```
File  Actions  Edit  View  Help
 GNU nano 8.4                                                task3_zombie_orphan.py
import os
import time

def create_zombie():
    pid = os.fork()
    if pid == 0:
        # Child sleeps briefly and exits
        print(f"Zombie Child: PID={os.getpid()} exiting...")
        os._exit(0)
    else:
        print(f"Parent PID={os.getpid()} not waiting for child {pid}")
        time.sleep(10)  # Gives time to check zombie with `ps -el | grep defunct`

def create_orphan():
    pid = os.fork()
    if pid == 0:
        time.sleep(5)
        print(f"Orphan Child: PID={os.getpid()}, new Parent PID={os.getppid()}")
        os._exit(0)
    else:
        print(f"Parent PID={os.getpid()} exiting immediately")
        os._exit(0)

if __name__ == "__main__":
    print("Creating zombie process...")
    create_zombie()
    time.sleep(2)
    print("\nCreating orphan process...")
    create_orphan()
```

OUTPUT

```
└─$ python3 task3_zombie_orphan.py
Creating zombie process...
Parent PID=18060 not waiting for child 18061
Zombie Child: PID=18061 exiting...

Creating orphan process...
Parent PID=18060 exiting immediately
```

```
└─$ Orphan Child: PID=18159, new Parent PID=1
ps -ps -el | grepunct
```

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

CODE

OUTPUT

Process Prioritization

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

Task 5:

CODE

```
File  Actions  Edit  View  Help
  GNU nano 8.4                                                    task5_priority.py
import os
import time

def cpu_intensive_task():
    count = 0
    for i in range(10**7):
        count += i
    print(f"Process PID={os.getpid()} finished counting.")

def main():
    nice_values = [0, 5, 10]  # Different priorities
    children_pids = []

    for nice_val in nice_values:
        pid = os.fork()
        if pid == 0:
            os.nice(nice_val)  # Set process priority
            print(f"Child PID={os.getpid()} with nice={nice_val} starting task... ")
            cpu_intensive_task()
            os._exit(0)
        else:
            children_pids.append(pid)

    # Parent waits
    for _ in children_pids:
        os.wait()

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

OUTPUT

```
└─$ python task5_priority.py
Child PID=27411 with nice=0 starting task...
Child PID=27412 with nice=5 starting task...
Child PID=27413 with nice=10 starting task...
Process PID=27411 finished counting.
Process PID=27412 finished counting.
Process PID=27413 finished counting.
```