OPERATING SYSTEMS

Lab Assignment Sheet-1

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().

INPUT-

```
import os
import sys
def main():
    try:
        N = int(input("Enter thr number of child processes: "))
    except ValueError:
        print("Please enter a valid integer.")
        sys.exit(1)
    print(f"Parent process PID: {os.getpid()} creating {N} child processes...\n")
    for i in range(N):
        pid = os.fork()
        if pid == 0:
            print(f"Child {i+1}: PID = {os.getpid()}, Parent PID = {os.getppid()}, Message = Hello from chld {i+1}")
            os._exit(0)
        else:
            continue
    for i in range(N):
        pid, status = os.wait()
        print(f"Parent: Child with PID = {pid} finished with status{status}")
if __name__ == "__main__":
    main()
```

OUTPUT

```
Enter thr number of child processes: 3
Parent process PID: 10039 creating 3 child processes...

Child 1: PID = 10061, Parent PID = 10039, Message = Hello from chld 1
Child 2: PID = 10064, Parent PID = 10039, Message = Hello from chld 2
Parent: Child with PID = 10061 finished with status0
Child 3: PID = 10065, Parent PID = 10039, Message = Hello from chld 3
Parent: Child with PID = 10064 finished with status0
Parent: Child with PID = 10065 finished with status0
```

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().

INPUT

```python
import os
import sys

def create_children_with_exec(n, command):
    children_pids = []

    for i in range(n):
        try:
            pid = os.fork()
        except OSError as e:
            print(f"Fork failed: {e}", file=sys.stderr)
            sys.exit(1)

        if pid == 0:
            print(f"\n[Child {i+1}] PID={os.getpid()}, Parent PID={os.getppid()}, executing command: {' '.join(command)}\n")
            try:
                os.execvp(command[0], command)
            except FileNotFoundError:
                print(f"Command not found: {command[0]}", file=sys.stderr)
                os._exit(1)
        else:
            children_pids.append(pid)

    for _ in children_pids:
        pid, status = os.wait()
        if os.WIFEXITED(status):
            print(f"[Parent] Child PID={pid} exited with status {os.WEXITSTATUS(status)}")
        else:
            print(f"[Parent] Child PID={pid} terminated abnormally")
```

```python
def main():
    try:
        n = int(input("Enter the number of child processes: "))
        if n <= 0:
            print("Number of processes must be positive.")
            return
    except ValueError:
        print("Invalid input. Please enter an integer.")
        return
    command = input("Enter the Linux command to execute (e.g., 'ls -l'): ").split()

    print(f"\n[Parent] PID={os.getpid()} creating {n} children to run: {' '.join(command)}\n")
    create_children_with_exec(n, command)

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

OUTPUT

```
Enter the number of child processes: 4
Enter the Linux command to execute (e.g., 'ls -l'): ls
[Parent] PID=13071 creating 4 children to run: ls
[Child 1] PID=13136, Parent PID=13071, executing command: ls
[Child 2] PID=13137, Parent PID=13071, executing command: ls
[Child 3] PID=13138, Parent PID=13071, executing command: ls

Desktop    even.py   forky.py               Pictures   Videos
Documents  fork1.py  Music                  Public
Downloads  fork2.py  os_alltasks_submission  Templates

[Child 4] PID=13139, Parent PID=13071, executing command: ls
[Parent] Child PID=13136 exited with status 0
Desktop    even.py   forky.py               Pictures   Videos
Documents  fork1.py  Music                  Public
Downloads  fork2.py  os_alltasks_submission  Templates

[Parent] Child PID=13137 exited with status 0
Desktop    even.py   forky.py               Pictures   Videos
Documents  fork1.py  Music                  Public
Downloads  fork2.py  os_alltasks_submission  Templates
[Parent] Child PID=13138 exited with status 0
Desktop    even.py   forky.py               Pictures   Videos
Documents  fork1.py  Music                  Public
Downloads  fork2.py  os_alltasks_submission  Templates
[Parent] Child PID=13139 exited with status 0
```

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.

INPUT

```
import os
import time
def zombie_process():
    pid = os.fork()
    if pid == 0:
        print(f"[Child] PID={os.getpid()}, Parent PID={os.getppid()} -> Exiting now.")
        os._exit(0)
    else:
        print(f"[Parent] PID={os.getpid()}, created child PID={pid}")
        print("[Parent] Not calling wait(), sleeping... Run 'ps -el | grep defunct' to see zombie.")
        time.sleep(30)
def orphan_process():
    pid = os.fork()
    if pid == 0:
        print(f"[Child] PID={os.getpid()}, Parent PID={os.getppid()} -> Sleeping...")
        time.sleep(20)
        print(f"[Child] PID={os.getpid()}, New Parent PID={os.getppid()} -> I am orphaned.")
    else:
        print(f"[Parent] PID={os.getpid()}, created child PID={pid} -> Exiting immediately.")
        os._exit(0)
if __name__ == "__main__":
    print("\n=== Task 3: Zombie & Orphan Processes ===")
    print("1. Zombie process demo")
    print("2. Orphan process demo")
    choice = input("Enter choice: ")
    if choice == "1":
        zombie_process()
    elif choice == "2":
        orphan_process()
    else:
        print("Invalid choice.")
```

OUTPUT

```
=== Task 3: Zombie & Orphan Processes ===
1. Zombie process demo
2. Orphan process demo
Enter choice: 1
[Parent] PID=1909, created child PID=1990
[Parent] Not calling wait(), sleeping... Run 'ps -el | grep defunct' to see zombie.
[Child] PID=1990, Parent PID=1909 -> Exiting now.
```

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

INPUT

```
import os
def inspect_process(pid):
    status_file = f"/proc/{pid}/status"
    exe_file = f"/proc/{pid}/exe"
    fd_dir = f"/proc/{pid}/fd"
    try:
        with open(status_file, "r") as f:
            name, state, vm_size = None, None, None
            for line in f:
                if line.startswith("Name:"):
                    name = line.split()[1]
                elif line.startswith("State:"):
                    state = " ".join(line.split()[1:])
                elif line.startswith("VmSize:"):
                    vm_size = " ".join(line.split()[1:])
            print(f"Process Name : {name}")
            print(f"Process State: {state}")
            print(f"Memory Usage : {vm_size}")
        try:
            exe_path = os.readlink(exe_file)
            print(f"Executable   : {exe_path}")
        except FileNotFoundError:
            print("Executable   : [Not available]")
```

```
        try:
            fds = os.listdir(fd_dir)
            print(f"Open FDs     : {len(fds)}")
            for fd in fds:
                try:
                    target = os.readlink(os.path.join(fd_dir, fd))
                    print(f"  FD {fd} -> {target}")
                except OSError:
                    print(f"  FD {fd} -> [unavailable]")
        except FileNotFoundError:
            print("Open FDs     : [Not available]")

    except FileNotFoundError:
        print(f"Process with PID {pid} does not exist.")
if __name__ == "__main__":
    pid = input("Enter PID to inspect: ")
    if pid.isdigit():
        inspect_process(pid)
    else:
        print("Invalid PID")
```

OUTPUT

```
Enter PID to inspect: 8165
Process Name : systemd-udevd
Process State: S (sleeping)
Memory Usage : 35684 kB
```

Task 5: Process Prioritization

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

INPUT

```python
import os
import time
import multiprocessing as mp

def cpu_worker(nice_value, duration):
    try:
        new_nice = os.nice(nice_value)
    except Exception as e:
        print(f"PID={os.getpid()} | Error setting nice({nice_value}): {e}")
        return

    pid = os.getpid()
    start = time.time()
    count = 0
    while time.time() - start < duration:
        count += 1

    print(f"PID={pid} | Requested nice={nice_value} | "
          f"Actual nice={new_nice} | Iterations={count}")

if __name__ == "__main__":
    duration = 5
    nice_values = [0, 5, 10, 15]
    procs = []

    for n in nice_values:
        p = mp.Process(target=cpu_worker, args=(n, duration))
        p.start()
        procs.append(p)

    for p in procs:
        p.join()
```

OUTPUT

```
PID=76505 | Requested nice=0 | Actual nice=0 | Iterations=12944342
PID=76506 | Requested nice=5 | Actual nice=5 | Iterations=9200200
PID=76507 | Requested nice=10 | Actual nice=10 | Iterations=3130143
PID=76508 | Requested nice=15 | Actual nice=15 | Iterations=1047727
```