Name – Nidhi Kumari

Roll No – 2401011482

Lab Experiment Sheet-1

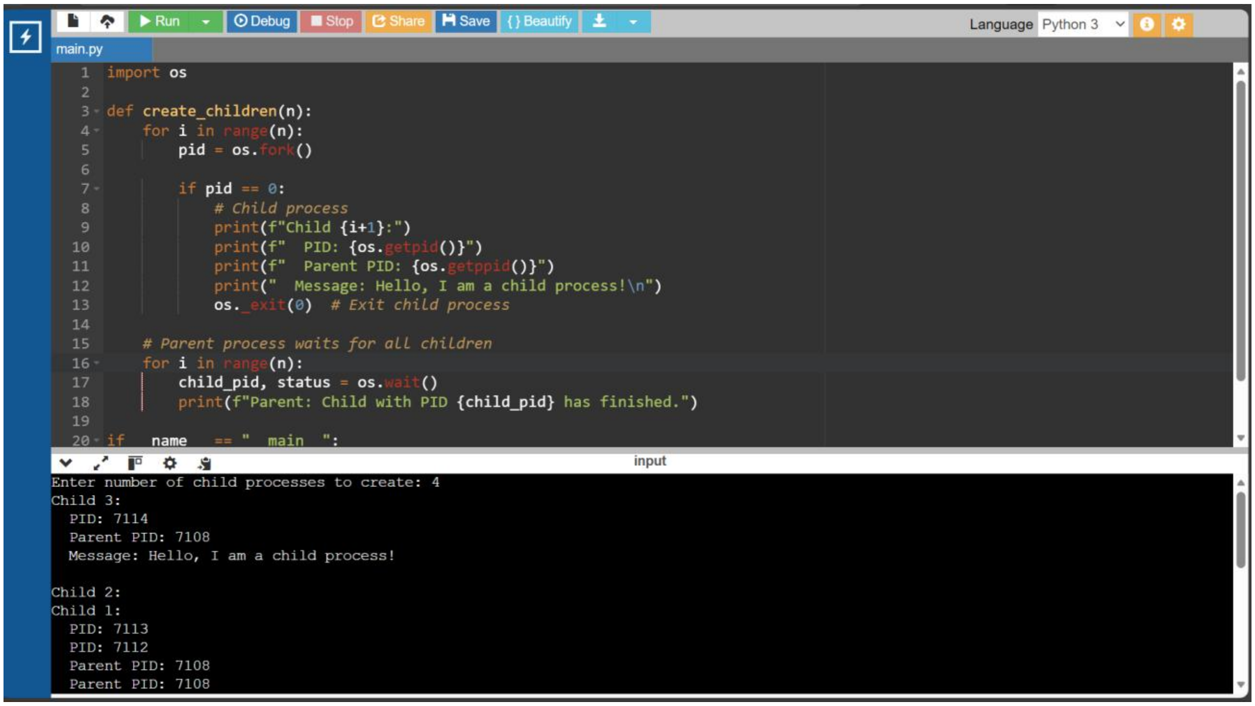
Task 1: Process Creation Utility

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 (PYTHON):

```
import os


def create_children(n):
    for i in range(n):
        pid = os.fork()


        if pid == 0
            print(f"Child {i+1}:")
            print(f"  PID: {os.getpid()}")
            print(f"  Parent PID: {os.getppid()}")
            print("  Message: Hello, I am a child process!\n")
            os._exit(0)  # Exit child process
    for i in range(n):
        child_pid, status = os.wait()
        print(f"Parent: Child with PID {child_pid} has finished.")

if __name__ == "__main__":
    N = int(input("Enter number of child processes to create: "))
    create_children(N)
```

OUTPUT:



```
import os

def create_children(n):
    for i in range(n):
        pid = os.fork()

        if pid == 0:
            # Child process
            print(f"Child {i+1}:")
            print(f"  PID: {os.getpid()}")
            print(f"  Parent PID: {os.getppid()}")
            print("  Message: Hello, I am a child process!\n")
            os._exit(0)  # Exit child process

    # Parent process waits for all children
    for i in range(n):
        child_pid, status = os.wait()
        print(f"Parent: Child with PID {child_pid} has finished.")

if __name__ == "__main__":
```

```
Enter number of child processes to create: 4
Child 3:
  PID: 7114
  Parent PID: 7108
  Message: Hello, I am a child process!

Child 2:
Child 1:
  PID: 7113
  PID: 7112
  Parent PID: 7108
  Parent PID: 7108
```

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()

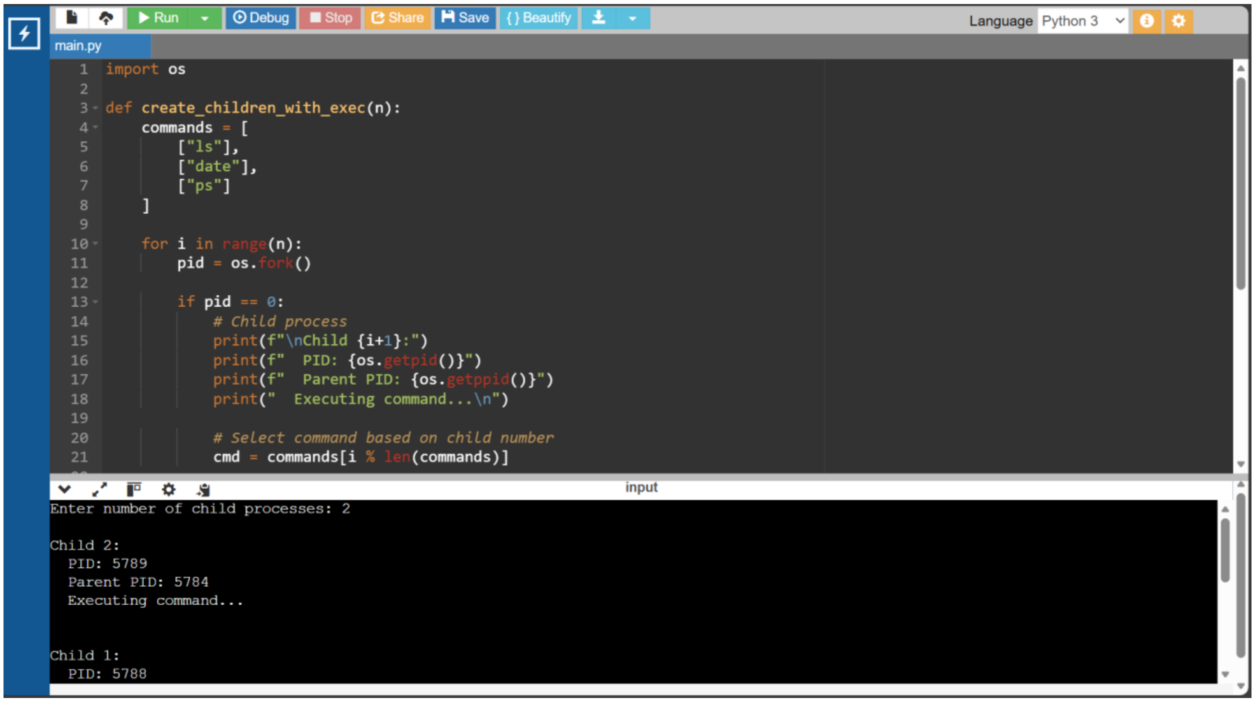
or

subprocess.run().

CODE(PYTHON):

```
import os


def create_children_with_exec(n):
    commands = [
        ["ls"],
        ["date"],
        ["ps"]
    ]

    for i in range(n):
        pid = os.fork()

        if pid == 0:
            # Child process
            print(f"\nChild {i+1}:")
            print(f"  PID: {os.getpid()}")
            print(f"  Parent PID: {os.getppid()}")
            print("  Executing command...\n
            cmd = commands[i % len(commands)]


            os.execvp(cmd[0], cmd)


            print("exec failed!")

            os._exit(1)

    for i in range(n):

        child_pid, status = os.wait()

        print(f"\nParent: Child with PID {child_pid} finished.")


if __name__ == "__main__":

    N = int(input("Enter number of child processes: "))

    create_children_with_exec(N)
```

OUTPUT:



```
import os

def create_children_with_exec(n):
    commands = [
        ["ls"],
        ["date"],
        ["ps"]
    ]

    for i in range(n):
        pid = os.fork()

        if pid == 0:
            # Child process
            print(f"\nChild {i+1}:")
            print(f"  PID: {os.getpid()}")
            print(f"  Parent PID: {os.getppid()}")
            print("  Executing command...\n")

            # Select command based on child number
            cmd = commands[i % len(commands)]
```

```
Enter number of child processes: 2

Child 2:
  PID: 5789
  Parent PID: 5784
  Executing command...


Child 1:
  PID: 5788
```

Task 3: Zombie & Orphan Processes

Zombie: Fork a child and skip wait() in the parent.

Orphan: Parent exits before the child finishes.

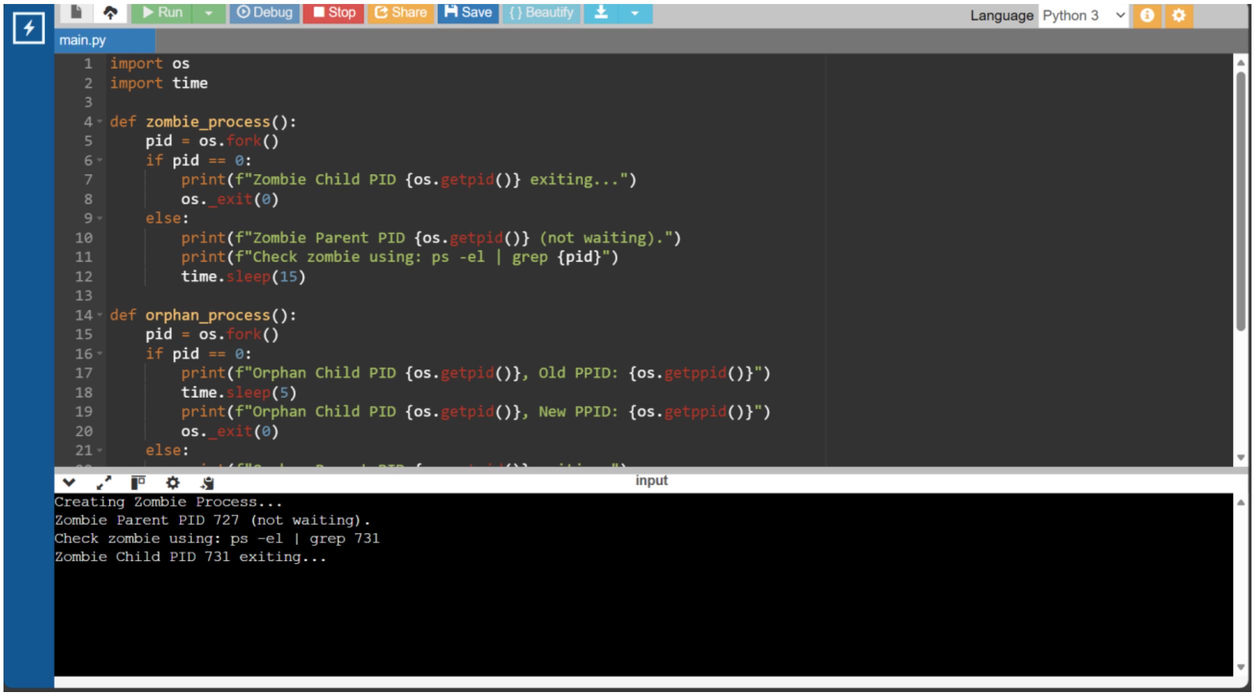
Use ps -el | grep defunct to identify zombies.

Code(python):

```python
import os
import time


def zombie_process():
  pid = os.fork()
  if pid == 0:
    print(f"Zombie Child PID {os.getpid()} exiting...")
    os._exit(0)
  else:
    print(f"Zombie Parent PID {os.getpid()} (not waiting).")
    print(f"Check zombie using: ps -el | grep {pid}")
    time.sleep(15)


def orphan_process():
  pid = os.fork()
  if pid == 0:
    print(f"Orphan Child PID {os.getpid()}, Old PPID: {os.getppid()}")
    time.sleep(5)
```

```
        print(f"Orphan Child PID {os.getpid()}, New PPID: {os.getppid()}")
        os._exit(0)
    else:
        print(f"Orphan Parent PID {os.getpid()} exiting.")
        os._exit(0)


if __name__ == "__main__":
    print("Creating Zombie Process...")
    zombie_process()
    time.sleep(2)
    print("\nCreating Orphan Process...")
    orphan_process()
```

OUTPUT:



```
import os
import time

def zombie_process():
    pid = os.fork()
    if pid == 0:
        print(f"Zombie Child PID {os.getpid()} exiting...")
        os._exit(0)
    else:
        print(f"Zombie Parent PID {os.getpid()} (not waiting).")
        print(f"Check zombie using: ps -el | grep {pid}")
        time.sleep(15)

def orphan_process():
    pid = os.fork()
    if pid == 0:
        print(f"Orphan Child PID {os.getpid()}, Old PPID: {os.getppid()}")
        time.sleep(5)
        print(f"Orphan Child PID {os.getpid()}, New PPID: {os.getppid()}")
        os._exit(0)
    else:
```

```
Creating Zombie Process...
Zombie Parent PID 727 (not waiting).
Check zombie using: ps -el | grep 731
Zombie Child PID 731 exiting...
```

Task 4: Inspecting Process Info from /proc

Take a PID as input. Read and print:

- Process name, state, memory usage from /proc/[pid]/status

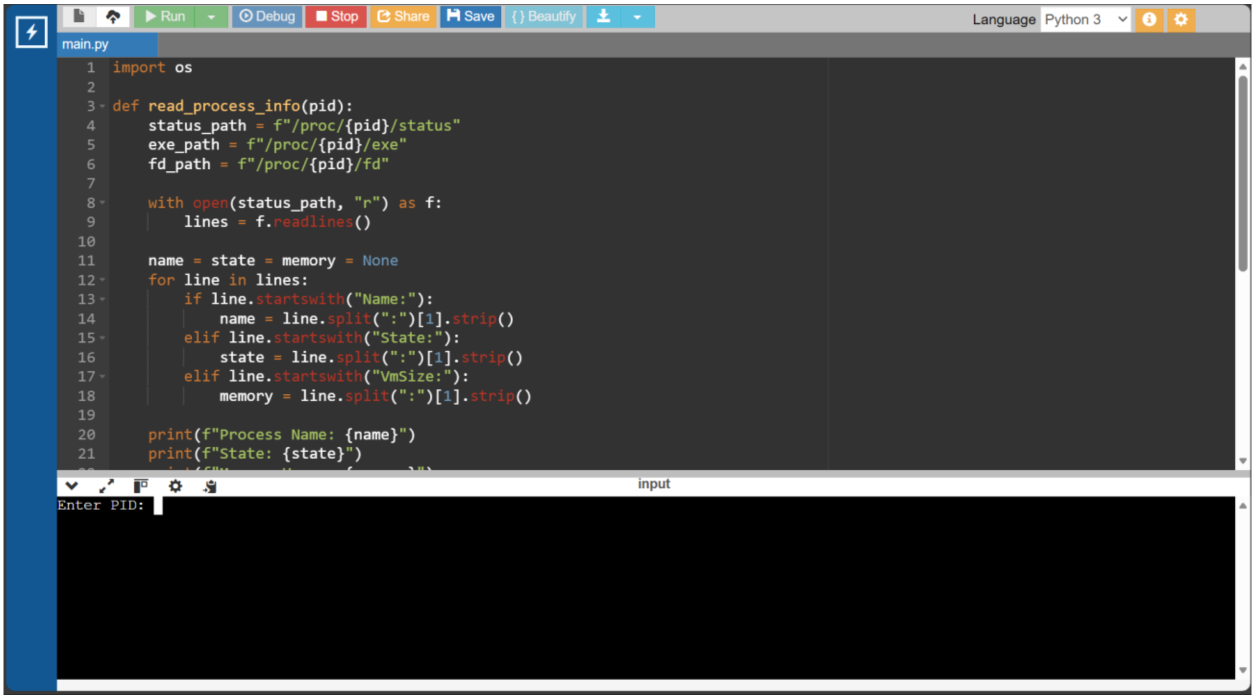
- Executable path from /proc/[pid]/exe

- Open file descriptors from /proc/[pid]/fd

CODE(PYTHON):

```
import os


def read_process_info(pid):
    status_path = f"/proc/{pid}/status"
    exe_path = f"/proc/{pid}/exe"
    fd_path = f"/proc/{pid}/fd"

    with open(status_path, "r") as f:
        lines = f.readlines()

    name = state = memory = None
    for line in lines:
        if line.startswith("Name:"):
            name = line.split(":")[1].strip()
        elif line.startswith("State:"):
            state = line.split(":")[1].strip()
        elif line.startswith("VmSize:"):
            memory = line.split(":")[1].strip()

    print(f"Process Name: {name}")
    print(f"State: {state}")
    print(f"Memory Usage: {memory}")

    try:
        exe = os.readlink(exe_path)
        print(f"Executable Path: {exe}")
    except:
        print("Executable Path: Not accessible")

    print("Open File Descriptors:")
    try:
        for fd in os.listdir(fd_path):
            link = os.readlink(os.path.join(fd_path, fd))
            print(f"  FD {fd} -> {link}")
    except:
        print("  Cannot access file descriptors")

if __name__ == "__main__":
    pid = input("Enter PID: ")
    read_process_info(pid)
```

OUTPUT:



```
import os

def read_process_info(pid):
    status_path = f"/proc/{pid}/status"
    exe_path = f"/proc/{pid}/exe"
    fd_path = f"/proc/{pid}/fd"

    with open(status_path, "r") as f:
        lines = f.readlines()

    name = state = memory = None
    for line in lines:
        if line.startswith("Name:"):
            name = line.split(":")[1].strip()
        elif line.startswith("State:"):
            state = line.split(":")[1].strip()
        elif line.startswith("VmSize:"):
            memory = line.split(":")[1].strip()

    print(f"Process Name: {name}")
    print(f"State: {state}")
```

```
Enter PID: 
```

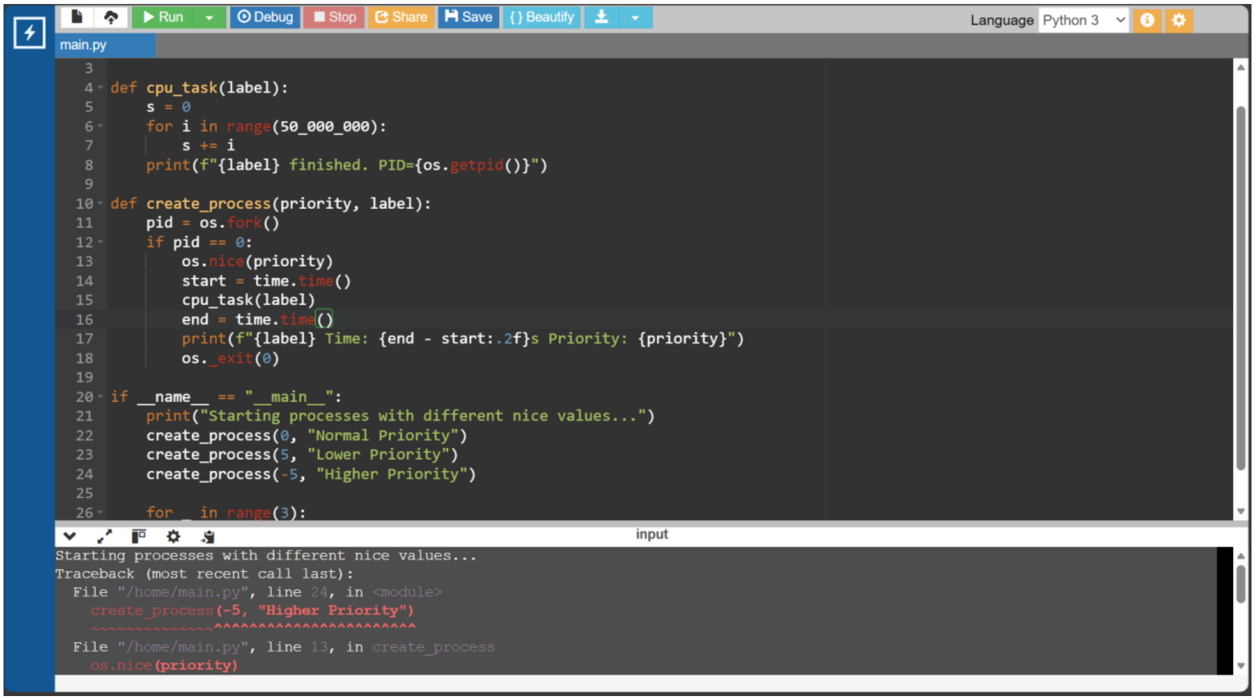
Task 5: Process Prioritization

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

CODE(PYTHON):

```
import os

import time


def cpu_task(label):

    s = 0

    for i in range(50_000_000):

        s += i
    print(f"{label} finished. PID={os.getpid()}")


def create_process(priority, label):
    pid = os.fork()
    if pid == 0:
        os.nice(priority)
        start = time.time()
        cpu_task(label)
        end = time.time()
        print(f"{label} Time: {end - start:.2f}s Priority: {priority}")
        os._exit(0)


if __name__ == "__main__":
    print("Starting processes with different nice values...")
    create_process(0, "Normal Priority")
    create_process(5, "Lower Priority")
    create_process(-5, "Higher Priority")

    for _ in range(3):
        os.wait()
```

OUTPUT:



```python

def cpu_task(label):
    s = 0
    for i in range(50_000_000):
        s += i
    print(f"{label} finished. PID={os.getpid()}")

def create_process(priority, label):
    pid = os.fork()
    if pid == 0:
        os.nice(priority)
        start = time.time()
        cpu_task(label)
        end = time.time()
        print(f"{label} Time: {end - start:.2f}s Priority: {priority}")
        os._exit(0)

if __name__ == "__main__":
    print("Starting processes with different nice values...")
    create_process(0, "Normal Priority")
    create_process(5, "Lower Priority")
    create_process(-5, "Higher Priority")

    for _ in range(3):
```

```
Starting processes with different nice values...
Traceback (most recent call last):
  File "/home/main.py", line 24, in <module>
    create_process(-5, "Higher Priority")
    ~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/main.py", line 13, in create_process
    os.nice(priority)
```