

**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:

The screenshot shows a Python code editor with a terminal window below it. The code in main.py creates four child processes. The terminal output shows the parent process waiting for children 1, 2, and 3, and then exiting.

```

main.py
1 import os
2
3 def create_children(n):
4     for i in range(n):
5         pid = os.fork()
6
7         if pid == 0:
8             # Child process
9             print(f"Child {i+1}:")
10            print(f"  PID: {os.getpid()}")
11            print(f"  Parent PID: {os.getppid()}")
12            print("  Message: Hello, I am a child process!\n")
13            os._exit(0) # Exit child process
14
15 # Parent process waits for all children
16 for i in range(n):
17     child_pid, status = os.wait()
18     print(f"Parent: Child with PID {child_pid} has finished.")
19
20 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.")

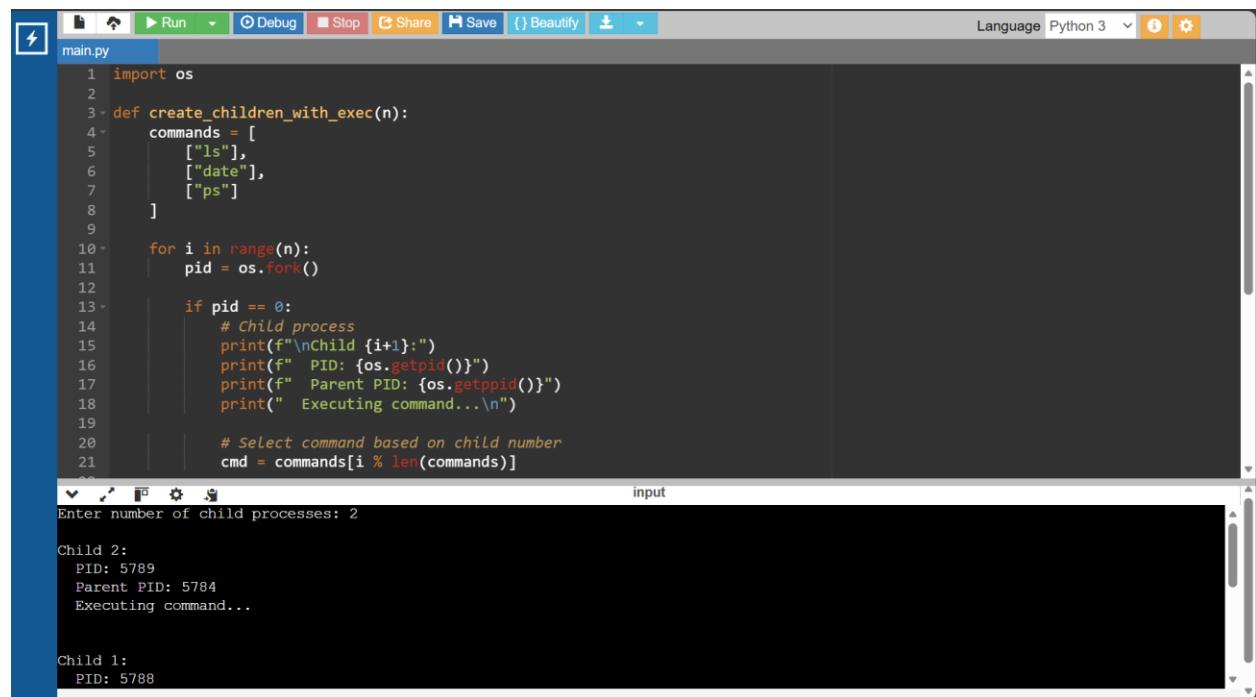
if __name__ == "__main__":

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

    create_children_with_exec(N)

```

## OUTPUT:



The screenshot shows a Python IDE interface with the file `main.py` open. The code implements a function `create_children_with_exec` that creates multiple child processes. Each child prints its PID, the PID of its parent, and the command it is executing. The user is prompted to enter the number of child processes, and the program outputs the results for each child.

```

import os
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")
    else:
        # Select command based on child number
        cmd = commands[i % len(commands)]

```

Input: Enter number of child processes: 2

Output:

```

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):

```
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:

The screenshot shows a Python IDE interface with the file `main.py` open. The code implements a zombie process and an orphan process using the `os.fork()` function. The zombie process exits immediately, while the parent continues to run and eventually becomes an orphan process. The output window displays the creation of the zombie process, its parent ID, and the command to check it using `ps -el | grep`. It also shows the creation of the orphan process, its PID, and its exit.

```

main.py
1 import os
2 import time
3
4 def zombie_process():
5     pid = os.fork()
6     if pid == 0:
7         print(f"Zombie Child PID {os.getpid()} exiting...")
8         os._exit(0)
9     else:
10        print(f"Zombie Parent PID {os.getpid()} (not waiting.)")
11        print(f"Check zombie using: ps -el | grep {pid}")
12        time.sleep(15)
13
14 def orphan_process():
15     pid = os.fork()
16     if pid == 0:
17         print(f"Orphan Child PID {os.getpid()}, Old PPID: {os.getppid()}")
18         time.sleep(5)
19         print(f"Orphan Child PID {os.getpid()}, New PPID: {os.getppid()}")
20         os._exit(0)
21     else:
22
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:

```
main.py
1 import os
2
3 def read_process_info(pid):
4     status_path = f"/proc/{pid}/status"
5     exe_path = f"/proc/{pid}/exe"
6     fd_path = f"/proc/{pid}/fd"
7
8     with open(status_path, "r") as f:
9         lines = f.readlines()
10
11    name = state = memory = None
12    for line in lines:
13        if line.startswith("Name:"):
14            name = line.split(":")[1].strip()
15        elif line.startswith("State:"):
16            state = line.split(":")[1].strip()
17        elif line.startswith("VmSize:"):
18            memory = line.split(":")[1].strip()
19
20    print(f"Process Name: {name}")
21    print(f"State: {state}")
```

input  
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()}\")\n\n\ndef create_process(priority, label):\n    pid = os.fork()\n\n    if pid == 0:\n        os.nice(priority)\n\n        start = time.time()\n\n        cpu_task(label)\n\n        end = time.time()\n\n        print(f"\{label} Time: {end - start:.2f}s Priority: {priority}\")\n\n        os._exit(0)\n\n\nif __name__ == "__main__":\n    print("Starting processes with different nice values...")\n\n    create_process(0, "Normal Priority")\n\n    create_process(5, "Lower Priority")\n\n    create_process(-5, "Higher Priority")\n\n\nfor _ in range(3):\n    os.wait()
```

## **OUTPUT:**