ASSIGNMENT – 1

Name: Erin Antil

Roll No.: 2301410011

Course: B.Tech 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:

File Actions Edit View Help

GNU nano 8.0 task1_fork.py

```
import os
import sys

def create_children(n):
    children = []

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

        if pid == 0:
            # Child process
            print(f"[Child] PID: {os.getpid()}, Parent PID: {os.getppid()}, Message: Hello, I am child {i+1}")
            os._exit(0)  # exit child safely
        else:
            # Parent process
            children.append(pid)

    # Parent waits for all children
    for pid in children:
        finished_pid, status = os.wait()
        print(f"[Parent] Child {finished_pid} terminated with status {status}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print(f"Usage: python3 {sys.argv[0]} <num_children>")
        sys.exit(1)

    n = int(sys.argv[1])
    create_children(n)
```

OUTPUT

```
┌──(kali㉿kali)-[~]
└─$ python3 task1_fork.py 3
[Child] PID: 7032, Parent PID: 7031, Message: Hello, I am child 1
[Parent] Child 7032 terminated with status 0
[Child] PID: 7034, Parent PID: 7031, Message: Hello, I am child 3
[Child] PID: 7033, Parent PID: 7031, Message: Hello, I am child 2
[Parent] Child 7034 terminated with status 0
[Parent] Child 7033 terminated with status 0
```

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:

```
File  Actions  Edit  View  Help
 GNU nano 8.0                                                task2_exec.py *
import os
import sys

def create_children(commands):
    children = []

    for i, cmd in enumerate(commands):
        pid = os.fork()

        if pid == 0:
            # Child process
            print(f"[Child] PID: {os.getpid()}, Parent PID: {os.getppid()}, Executing: {cmd}")
            try:
                # Replace this process with the given command
                os.execvp(cmd[0], cmd)
            except FileNotFoundError:
                print(f"Command not found: {cmd[0]}")
                os._exit(1)
        else:
            # Parent process
            children.append(pid)

    # Parent waits for all children
    for _ in children:
        finished_pid, status = os.wait()
        print(f"[Parent] Child {finished_pid} terminated with status {status}")

if __name__ == "__main__":
    # Example commands (you can edit this list)
    commands = [
        ["ls", "-l"],
        ["date"],
        ["ps", "-f"]
    ]
    create_children(commands)
```

OUTPUT

```
┌──(kali㉿kali)-[~]
└─$ python3 task2_exec.py

[Child] PID: 9237, Parent PID: 9236, Executing: ['ls', '-l']
[Child] PID: 9238, Parent PID: 9236, Executing: ['date']
[Child] PID: 9239, Parent PID: 9236, Executing: ['ps', '-f']
Mon Oct  6 12:20:14 AM EDT 2025
[Parent] Child 9238 terminated with status 0
total 44
drwxr-xr-x 2 kali kali 4096 Aug 21  2024 Desktop
drwxr-xr-x 2 kali kali 4096 Aug 21  2024 Documents
drwxr-xr-x 2 kali kali 4096 Aug 22  2024 Downloads
drwxr-xr-x 2 kali kali 4096 Aug 21  2024 Music
drwxr-xr-x 2 kali kali 4096 Oct  6 00:13 Pictures
drwxr-xr-x 2 kali kali 4096 Aug 21  2024 Public
-rw-rw-r-- 1 kali kali  763 Oct  6 00:15 task1_fork.py
-rw------- 1 kali kali  937 Oct  6 00:09 task1_fork.py.save
-rw-rw-r-- 1 kali kali  965 Oct  6 00:20 task2_exec.py
drwxr-xr-x 2 kali kali 4096 Aug 21  2024 Templates
drwxr-xr-x 2 kali kali 4096 Aug 21  2024 Videos
[Parent] Child 9237 terminated with status 0
UID          PID    PPID  C STIME TTY          TIME CMD
kali        7299    7288  0 00:16 pts/3    00:00:00 /usr/bin/zsh
kali        9236    7299 55 00:20 pts/3    00:00:00 python3 task2_exec.py
kali        9239    9236 99 00:20 pts/3    00:00:00 ps -f
[Parent] Child 9239 terminated 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.

CODE

```
File  Actions  Edit  View  Help
 GNU nano 8.0                                    task3_zombie_orphan.py *
import os
import time

def create_zombie():
    print("\n── Zombie Process Demo ──")
    pid = os.fork()

    if pid == 0:
        # Child process: exit immediately
        print(f"[Zombie Child] PID={os.getpid()}, Parent PID={os.getppid()} exiting...")
        os._exit(0)
    else:
        # Parent does NOT call wait()
        print(f"[Parent] PID={os.getpid()} created child {pid} (will become zombie)")
        print("Run: ps -el | grep defunct   to see the zombie process")
        time.sleep(15)  # keep parent alive so you can inspect

def create_orphan():
    print("\n── Orphan Process Demo ──")
    pid = os.fork()

    if pid == 0:
        # Child: sleep longer than parent so it outlives it
        print(f"[Orphan Child] PID={os.getpid()}, Parent PID={os.getppid()} (before)")
        time.sleep(10)
        print(f"[Orphan Child] PID={os.getpid()}, New Parent PID={os.getppid()} (after parent exited)")
        os._exit(0)
    else:
        # Parent exits immediately
        print(f"[Parent] PID={os.getpid()} created child {pid} and now exiting...")
        os._exit(0)


if __name__ == "__main__":
    print("Choose demo:")
    print("1. Zombie Process")
    print("2. Orphan Process")

    choice = input("Enter 1 or 2: ").strip()
```

```
    choice = input("Enter 1 or 2: ").strip()

    if choice == "1":
        create_zombie()
    elif choice == "2":
        create_orphan()
    else:
        print("Invalid choice")
```

OUTPUT

```
┌──(kali㉿kali)-[~]
└─$ python3 task3_zombie_orphan.py

Choose demo:
1. Zombie Process
2. Orphan Process
Enter 1 or 2: 1

── Zombie Process Demo ──
[Parent] PID=13195 created child 13604 (will become zombie)
Run: ps -el | grep defunct   to see the zombie process
[Zombie Child] PID=13604, Parent PID=13195 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

```
File  Actions  Edit  View  Help
 GNU nano 8.0                                  task4_proc_inspect.py *
import os
import sys

def inspect_process(pid):
    status_file = f"/proc/{pid}/status"
    exe_file = f"/proc/{pid}/exe"
    fd_dir = f"/proc/{pid}/fd"

    try:
        # ── Process status info ──
        with open(status_file, "r") as f:
            status_lines = f.readlines()

        # Extract specific fields
        process_name = None
        state = None
        memory = None

        for line in status_lines:
            if line.startswith("Name:"):
                process_name = line.split()[1]
            elif line.startswith("State:"):
                state = " ".join(line.split()[1:])
            elif line.startswith("VmRSS:"):
                memory = " ".join(line.split()[1:])

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

        # ── Executable path ──
        try:
            exe_path = os.readlink(exe_file)
            print(f"Executable   : {exe_path}")
        except FileNotFoundError:
            print("Executable   : [No exe file found]")

        # ── Open file descriptors ──
        print("\nOpen File Descriptors:")
        if os.path.exists(fd_dir):
            for fd in os.listdir(fd_dir):
```

```
        if os.path.exists(fd_dir):
            for fd in os.listdir(fd_dir):
                try:
                    target = os.readlink(os.path.join(fd_dir, fd))
                    print(f"  FD {fd} → {target}")
                except OSError:
                    print(f"  FD {fd} → [unreadable]")
        else:
            print("  [No fd directory found]")

    except FileNotFoundError:
        print(f"Process {pid} not found.")


if __name__ == "__main__":
    if len(sys.argv) != 2:
        print(f"Usage: python3 {sys.argv[0]} <pid>")
        sys.exit(1)

    pid = sys.argv[1]
    inspect_process(pid)
```

OUTPUT

```
┌──(kali㉿kali)-[~]
└─$ sudo python3 task4_proc_inspect.py 1

Process Name : systemd
State        : S (sleeping)
Memory Usage : 12872 kB
Executable   : /usr/lib/systemd/systemd

Open File Descriptors:
  FD 0 → /dev/null
  FD 1 → /dev/null
  FD 2 → /dev/null
  FD 3 → /dev/kmsg
  FD 4 → anon_inode:[eventpoll]
  FD 5 → anon_inode:[signalfd]
  FD 6 → anon_inode:inotify
  FD 7 → /sys/fs/cgroup
  FD 8 → anon_inode:[timerfd]
  FD 9 → /usr/lib/systemd/systemd-executor
  FD 10 → socket:[15218]
  FD 11 → anon_inode:inotify
  FD 12 → anon_inode:inotify
  FD 13 → anon_inode:[eventpoll]
  FD 14 → /proc/1/mountinfo
  FD 15 → anon_inode:inotify
  FD 16 → /proc/swaps
  FD 17 → socket:[1392]
  FD 18 → socket:[1393]
  FD 19 → socket:[1394]
  FD 20 → socket:[1395]
  FD 21 → socket:[1396]
  FD 22 → socket:[1398]
  FD 23 → socket:[1399]
  FD 24 → socket:[15219]
  FD 28 → socket:[1404]
  FD 30 → /dev/autofs
  FD 31 → pipe:[6070]
  FD 32 → socket:[6073]
  FD 33 → /run/initctl
  FD 34 → socket:[6074]
  FD 35 → socket:[6076]
```

```
  FD 35 → socket:[6076]
  FD 36 → socket:[6078]
  FD 37 → socket:[6080]
  FD 38 → socket:[6082]
  FD 39 → anon_inode:[timerfd]
  FD 40 → socket:[6132]
  FD 41 → socket:[2669]
  FD 43 → anon_inode:[pidfd]
  FD 44 → socket:[2724]
  FD 45 → anon_inode:inotify
  FD 46 → anon_inode:[pidfd]
  FD 47 → anon_inode:[pidfd]
  FD 48 → anon_inode:bpf-prog
  FD 49 → anon_inode:bpf-prog
  FD 50 → anon_inode:[pidfd]
  FD 51 → anon_inode:[pidfd]
  FD 52 → socket:[2938]
  FD 54 → anon_inode:bpf-prog
  FD 55 → socket:[1752]
  FD 56 → socket:[1753]
  FD 57 → anon_inode:[pidfd]
  FD 58 → anon_inode:[pidfd]
  FD 59 → anon_inode:[pidfd]
  FD 60 → anon_inode:[pidfd]
  FD 61 → anon_inode:[pidfd]
  FD 62 → socket:[7626]
  FD 63 → anon_inode:[timerfd]
  FD 64 → anon_inode:bpf-prog
  FD 65 → anon_inode:bpf-prog
  FD 66 → anon_inode:bpf-prog
  FD 67 → anon_inode:[pidfd]
  FD 68 → anon_inode:[pidfd]
  FD 69 → anon_inode:[pidfd]
  FD 70 → socket:[7601]
  FD 71 → socket:[4002]
  FD 72 → socket:[4004]
  FD 73 → anon_inode:bpf-prog
  FD 74 → socket:[4007]
  FD 75 → socket:[4008]
  FD 76 → anon_inode:[pidfd]
  FD 77 → anon_inode:[pidfd]
  FD 78 → anon_inode:[pidfd]
  FD 79 → anon_inode:[pidfd]
```

```
FD 79 → anon_inode:[pidfd]
FD 80 → socket:[10944]
FD 81 → socket:[8302]
FD 82 → socket:[10945]
FD 83 → socket:[10946]
FD 84 → socket:[3780]
FD 86 → anon_inode:bpf-prog
FD 87 → anon_inode:[pidfd]
FD 88 → anon_inode:[pidfd]
FD 89 → anon_inode:bpf-prog
FD 90 → anon_inode:bpf-prog
FD 91 → anon_inode:bpf-prog
FD 92 → anon_inode:[pidfd]
FD 93 → anon_inode:[pidfd]
FD 94 → socket:[4037]
FD 95 → socket:[4065]
FD 96 → socket:[4066]
FD 97 → /sys/fs/cgroup/init.scope/memory.pressure
FD 99 → /dev/rfkill
FD 102 → socket:[10947]
FD 103 → anon_inode:[pidfd]
FD 105 → anon_inode:[pidfd]
FD 110 → socket:[8264]
FD 112 → anon_inode:[pidfd]
FD 113 → anon_inode:[pidfd]
FD 114 → anon_inode:[pidfd]
FD 115 → socket:[8267]
FD 116 → anon_inode:[pidfd]
FD 117 → anon_inode:[pidfd]
FD 118 → socket:[9544]
FD 119 → socket:[10948]
FD 120 → socket:[9230]
FD 121 → socket:[10745]
FD 122 → socket:[10746]
FD 123 → anon_inode:[pidfd]
FD 124 → anon_inode:[pidfd]
FD 125 → socket:[8636]
FD 126 → socket:[10949]
FD 127 → socket:[11376]
FD 130 → socket:[9071]
FD 131 → socket:[9093]
FD 132 → socket:[9800]
FD 133 → socket:[9864]
```

```
FD 133 → socket:[9864]
FD 134 → socket:[12435]
FD 135 → socket:[11610]
FD 136 → anon_inode:bpf-prog
FD 137 → anon_inode:bpf-prog
FD 138 → anon_inode:[pidfd]
FD 139 → anon_inode:[pidfd]
FD 140 → socket:[11450]
FD 141 → socket:[11625]
FD 142 → anon_inode:[pidfd]
FD 143 → anon_inode:[pidfd]
FD 145 → socket:[9824]
FD 149 → socket:[13365]
FD 150 → socket:[13371]
FD 151 → anon_inode:[pidfd]
FD 152 → anon_inode:[pidfd]
FD 153 → socket:[12452]
FD 154 → socket:[13386]
FD 155 → socket:[13409]
```

Task 5: Process Prioritization

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

CODE

```
File  Actions  Edit  View  Help
 GNU nano 8.0                                                    task5_priority.py *
import os
import time

def cpu_intensive_task(name, duration=5):
    """Simulate a CPU-heavy task."""
    print(f"[{name}] PID={os.getpid()}, Parent={os.getppid()}, Starting task...")
    start_time = time.time()
    count = 0
    while time.time() - start_time < duration:
        count += 1  # Busy loop
    print(f"[{name}] PID={os.getpid()}, Finished task. Loop count={count}")

def create_children():
    # List of child processes with (name, nice_value)
    children_info = [
        ("HighPriority", -5),   # higher priority
        ("NormalPriority", 0),  # default
        ("LowPriority", 5)      # lower priority
    ]

    children = []

    for name, nice_value in children_info:
        pid = os.fork()

        if pid == 0:
            # Child process
            try:
                current_nice = os.nice(0)
                os.nice(nice_value - current_nice)  # set desired nice
                print(f"[{name}] PID={os.getpid()} Nice={os.nice(0)}")
            except PermissionError:
                print(f"[{name}] PID={os.getpid()} Unable to change nice value, running default.")

            cpu_intensive_task(name)
            os._exit(0)
        else:
            # Parent
            children.append(pid)

    # Wait for all children
    for _ in children:
        finished_pid, status = os.wait()
        print(f"[Parent] Child {finished_pid} terminated with status {status}")

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

OUTPUT

```
┌──(kali㉿kali)-[~]
└─$ sudo python3 task5_priority.py
[sudo] password for kali:
[HighPriority] PID=20005 Nice=-5
[HighPriority] PID=20005, Parent=20004, Starting task...
[NormalPriority] PID=20006 Nice=0
[NormalPriority] PID=20006, Parent=20004, Starting task...
[LowPriority] PID=20007 Nice=5
[LowPriority] PID=20007, Parent=20004, Starting task...
[HighPriority] PID=20005, Finished task. Loop count=11401790
[NormalPriority] PID=20006, Finished task. Loop count=10473856
[Parent] Child 20005 terminated with status 0
[Parent] Child 20006 terminated with status 0
[LowPriority] PID=20007, Finished task. Loop count=9922308
[Parent] Child 20007 terminated with status 0
```