ASSIGNMENT 4

Name: Payal Rathore

Roll No: 2301410022

Course: BTech CSE (Cyber Security)

Task 1: Batch Processing Simulation (Python)

Write a Python script to execute multiple .py files sequentially, mimicking batch processing.

CODE

```
File  Actions  Edit  View  Help
  GNU nano 8.4
import subprocess

# List of Python scripts to execute sequentially
scripts = ['script1.py', 'script2.py', 'script3.py']

for script in scripts:
    print(f"\n=== Executing {script} ===")
    result = subprocess.run(['python3', script])
    if result.returncode == 0:
        print(f"{script} executed successfully.\n")
    else:
        print(f"Error while executing {script}.\n")
```

OUTPUT

```
┌──(lakshita㉿kali)-[~/linux_process_simulation/Assignment4_BatchProcessing]
└─$ nano batch_runner.py

┌──(lakshita㉿kali)-[~/linux_process_simulation/Assignment4_BatchProcessing]
└─$ python3 batch_runner.py

=== Executing script1.py ===
Running Script 1
script1.py executed successfully.


=== Executing script2.py ===
Running Script 2
script2.py executed successfully.


=== Executing script3.py ===
Running Script 3
script3.py executed successfully.
```

Task 2: System Startup and Logging

Simulate system startup using Python by creating multiple processes and logging their start and end into a log file.

CODE

lakshita@kali: ~/linux_process_simulat

File Actions Edit View Help

GNU nano 8.4 startup_

```
#!/usr/bin/env python3
"""
startup_simulation.py
Simulate system startup: create multiple processes (multiprocessing) which run simple tasks,
and log lifecycle events (start, end, elapsed) into process_log.txt
"""
import logging
import multiprocessing as mp
import time
import os

LOGFILE = "process_log.txt"

def setup_logging():
    logging.basicConfig(
        filename=LOGFILE,
        level=logging.INFO,
        format="%(asctime)s [%(processName)s:%(process)d] %(levelname)s: %(message)s"
    )
    logging.getLogger().addHandler(logging.StreamHandler())  # also print to console

def worker(name, duration):
    logging.info(f"Worker {name} starting (PID={os.getpid()})")
    t0 = time.time()
    # simulate work
    time.sleep(duration)
    elapsed = time.time() - t0
    logging.info(f"Worker {name} finished (PID={os.getpid()}) elapsed={elapsed:.2f}s")

def main():
    setup_logging()
    logging.info("Startup simulation beginning")

    # Configure simulated startup tasks: (name, duration seconds)
    tasks = [
        ("init_services", 2),
        ("mount_filesystems", 1),
        ("network_manager", 3),
        ("login_manager", 1.5),
        ("cron_jobs", 0.8),
    ]
```

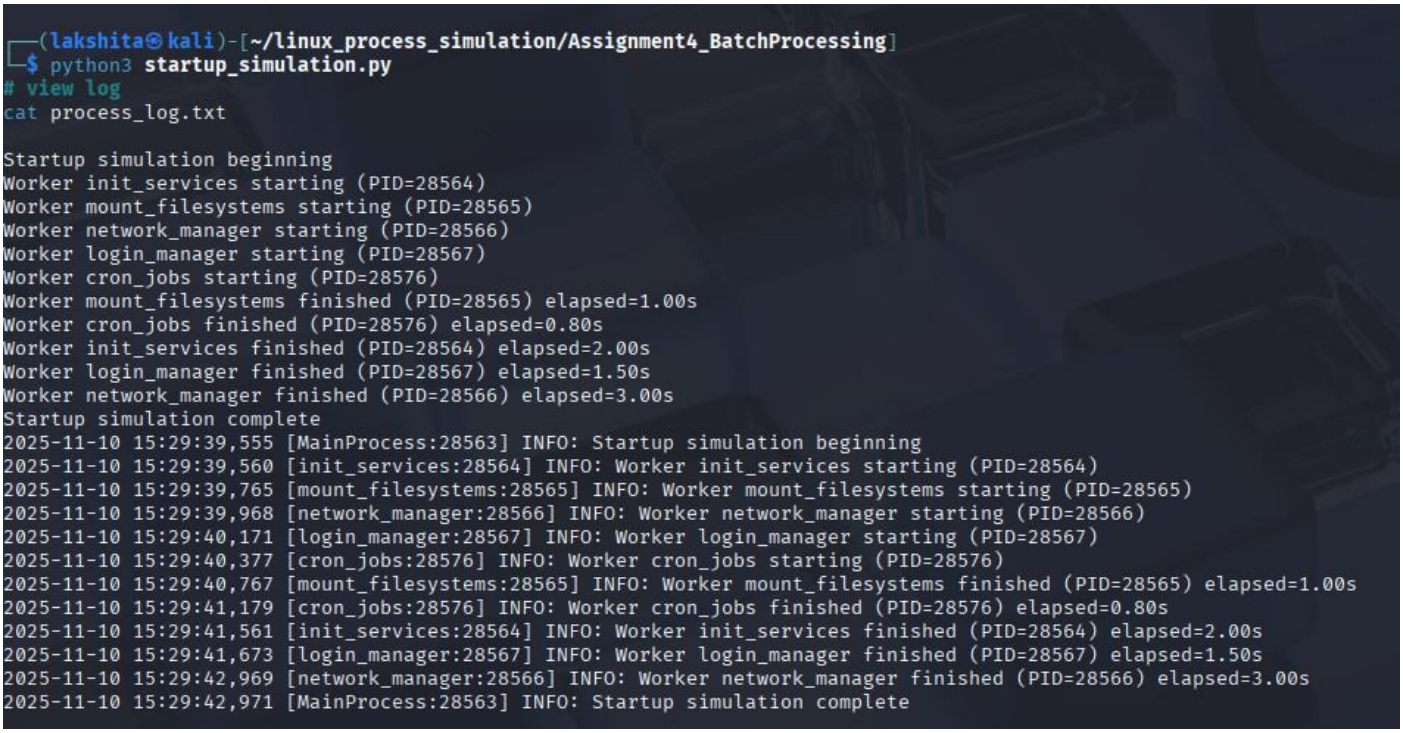
```
    procs = []
    for name, dur in tasks:
        p = mp.Process(target=worker, args=(name, dur), name=name)
        p.start()
        procs.append(p)
        # optional staggered start like real system
        time.sleep(0.2)

    # wait for completion
    for p in procs:
        p.join()

    logging.info("Startup simulation complete")

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

OUTPUT

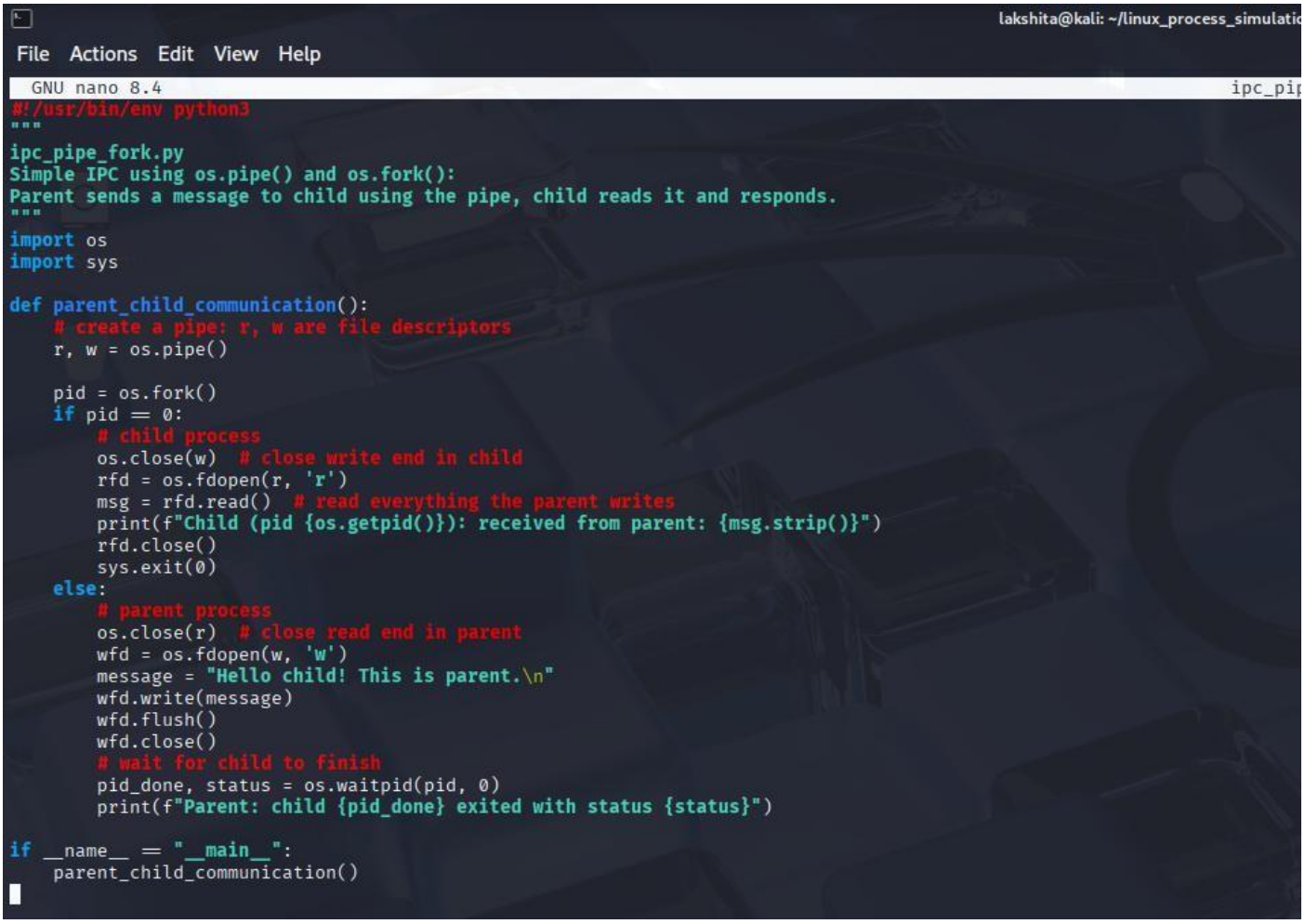

(lakshita㉿kali)-[~/linux_process_simulation/Assignment4_BatchProcessing]
$ python3 startup_simulation.py
# view log
cat process_log.txt
Startup simulation beginning
Worker init_services starting (PID=28564)
Worker mount_filesystems starting (PID=28565)
Worker network_manager starting (PID=28566)
Worker login_manager starting (PID=28567)
Worker cron_jobs starting (PID=28576)
Worker mount_filesystems finished (PID=28565) elapsed=1.00s
Worker cron_jobs finished (PID=28576) elapsed=0.80s
Worker init_services finished (PID=28564) elapsed=2.00s
Worker login_manager finished (PID=28567) elapsed=1.50s
Worker network_manager finished (PID=28566) elapsed=3.00s
Startup simulation complete
2025-11-10 15:29:39,555 [MainProcess:28563] INFO: Startup simulation beginning
2025-11-10 15:29:39,560 [init_services:28564] INFO: Worker init_services starting (PID=28564)
2025-11-10 15:29:39,765 [mount_filesystems:28565] INFO: Worker mount_filesystems starting (PID=28565)
2025-11-10 15:29:39,968 [network_manager:28566] INFO: Worker network_manager starting (PID=28566)
2025-11-10 15:29:40,171 [login_manager:28567] INFO: Worker login_manager starting (PID=28567)
2025-11-10 15:29:40,377 [cron_jobs:28576] INFO: Worker cron_jobs starting (PID=28576)
2025-11-10 15:29:40,767 [mount_filesystems:28565] INFO: Worker mount_filesystems finished (PID=28565) elapsed=1.00s
2025-11-10 15:29:41,179 [cron_jobs:28576] INFO: Worker cron_jobs finished (PID=28576) elapsed=0.80s
2025-11-10 15:29:41,561 [init_services:28564] INFO: Worker init_services finished (PID=28564) elapsed=2.00s
2025-11-10 15:29:41,673 [login_manager:28567] INFO: Worker login_manager finished (PID=28567) elapsed=1.50s
2025-11-10 15:29:42,969 [network_manager:28566] INFO: Worker network_manager finished (PID=28566) elapsed=3.00s
2025-11-10 15:29:42,971 [MainProcess:28563] INFO: Startup simulation complete

Task 3: System Calls and IPC (Python - fork, exec, pipe)

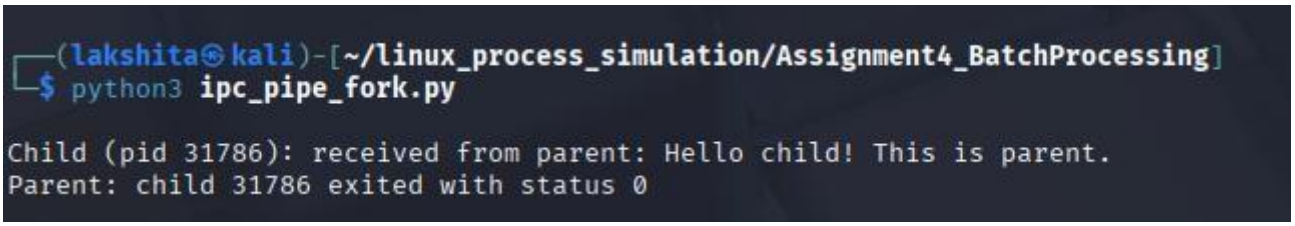
Use system calls (fork(), exec(), wait()) and implement basic Inter-Process Communication using pipes in C or Python.

- **ipc_pipe_fork.py :** parent and child communicate via an anonymous pipe (os.pipe + os.fork).

CODE

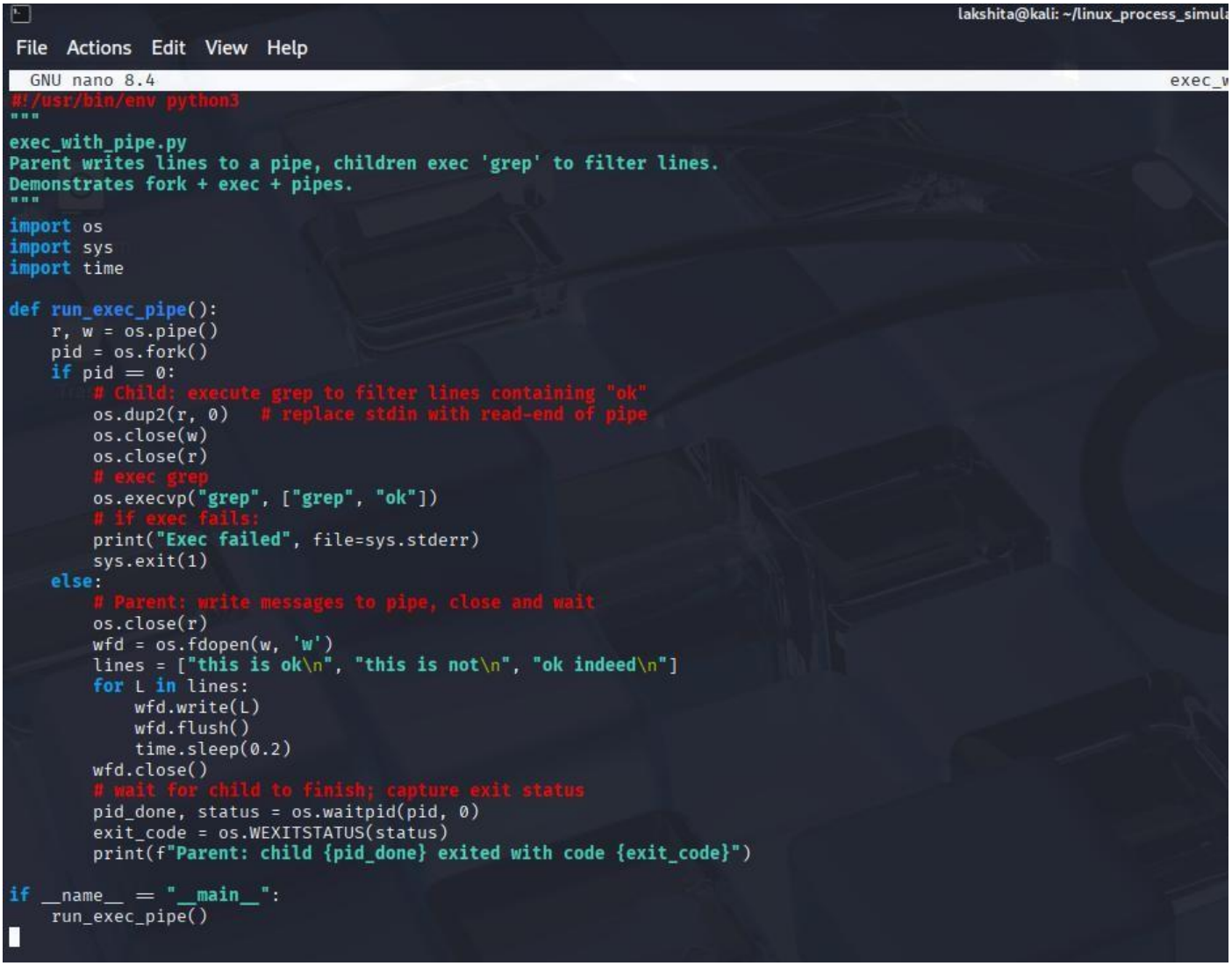


OUTPUT

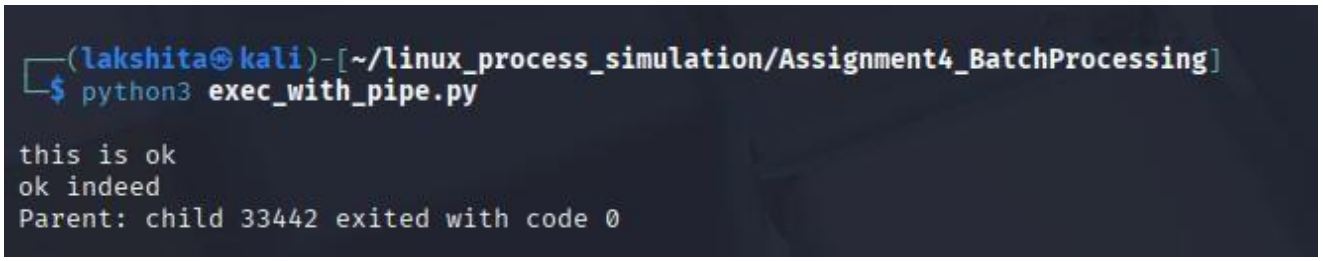


- **exec_with_pipe.py:** parent creates pipe, forks, child os.execvp() to run grep (or cat) and parent writes into pipe.

CODE



OUTPUT

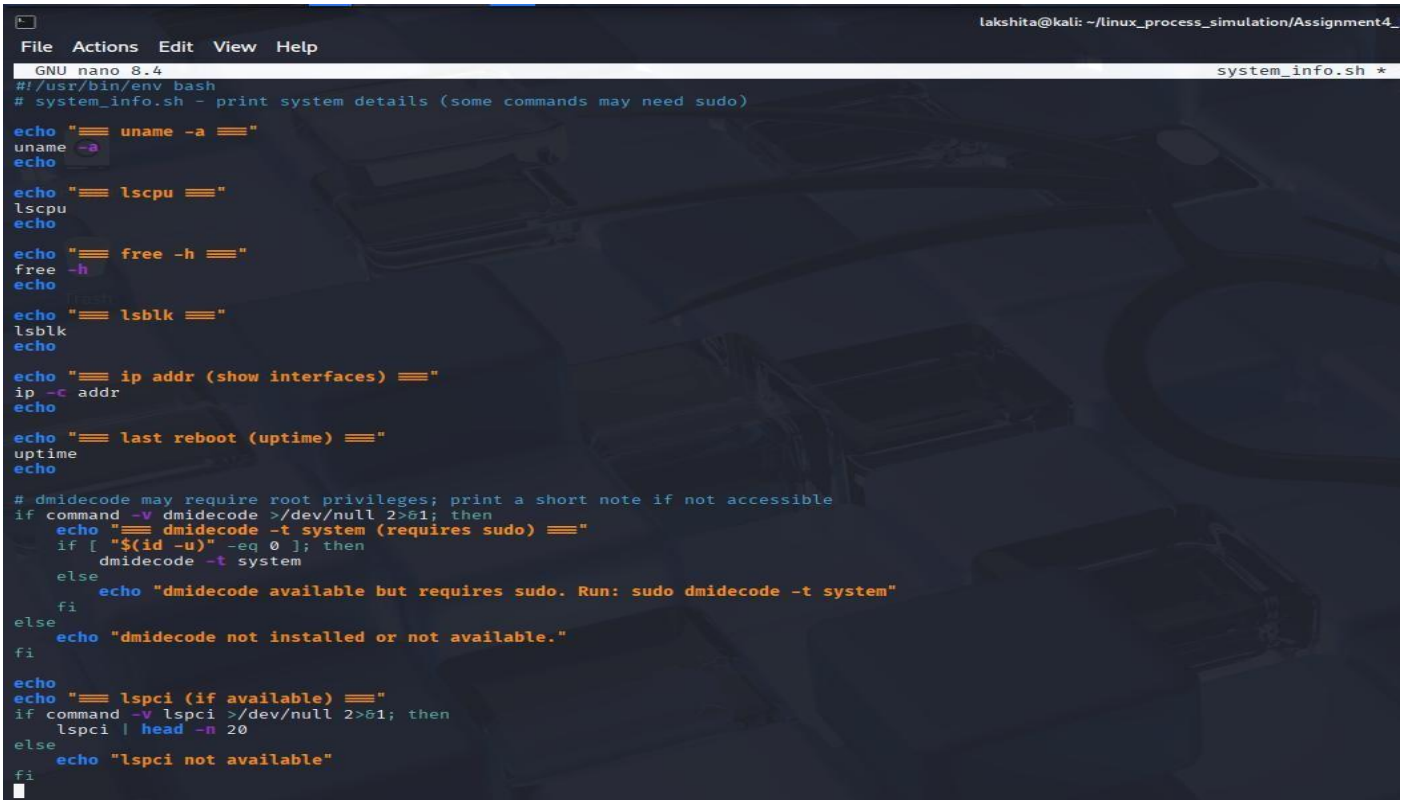


Task 4: VM Detection and Shell Interaction

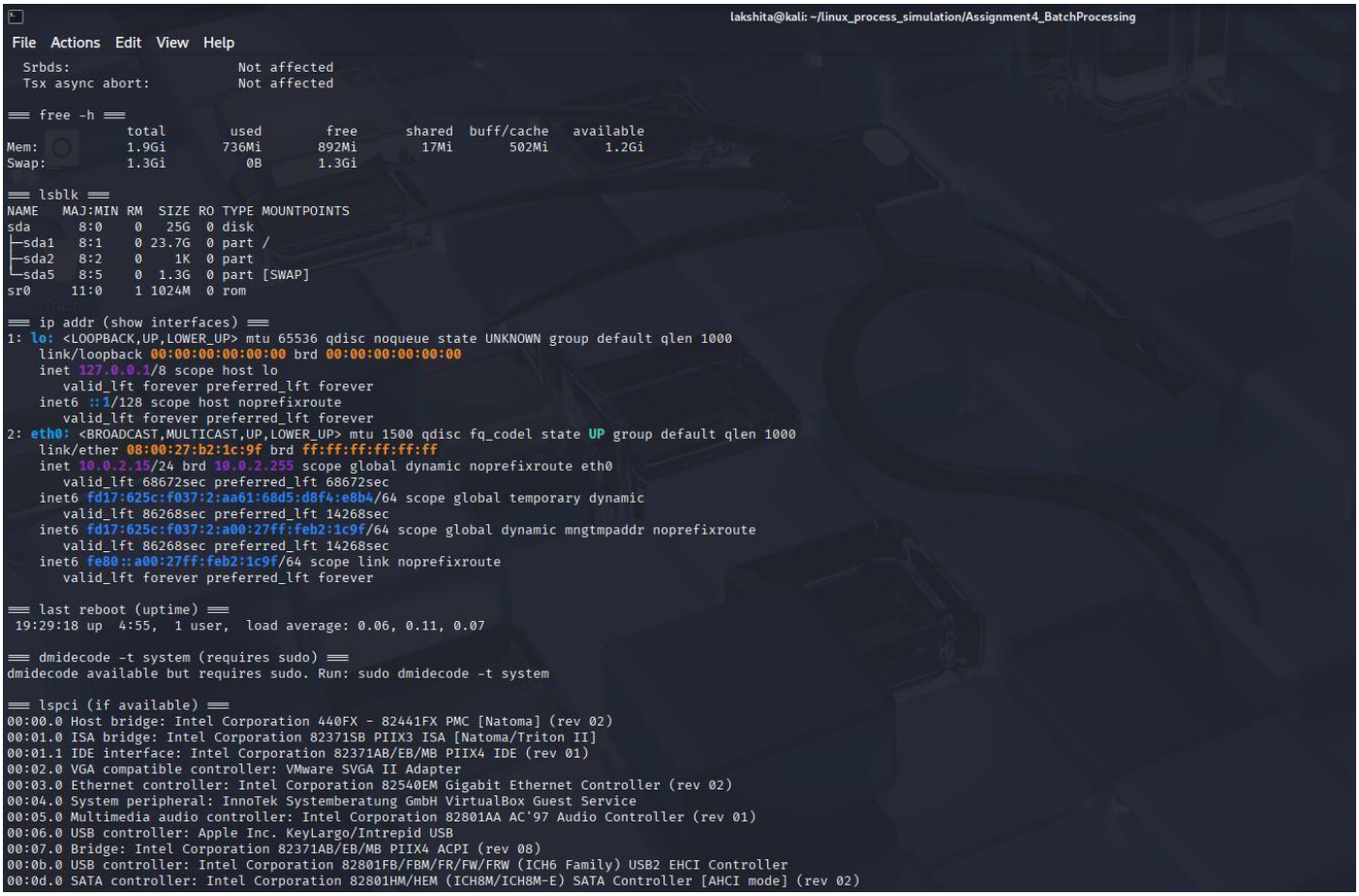
Create a shell script to print system details and a Python script to detect if the system is running inside a virtual machine.

- Shell script to print system details

CODE

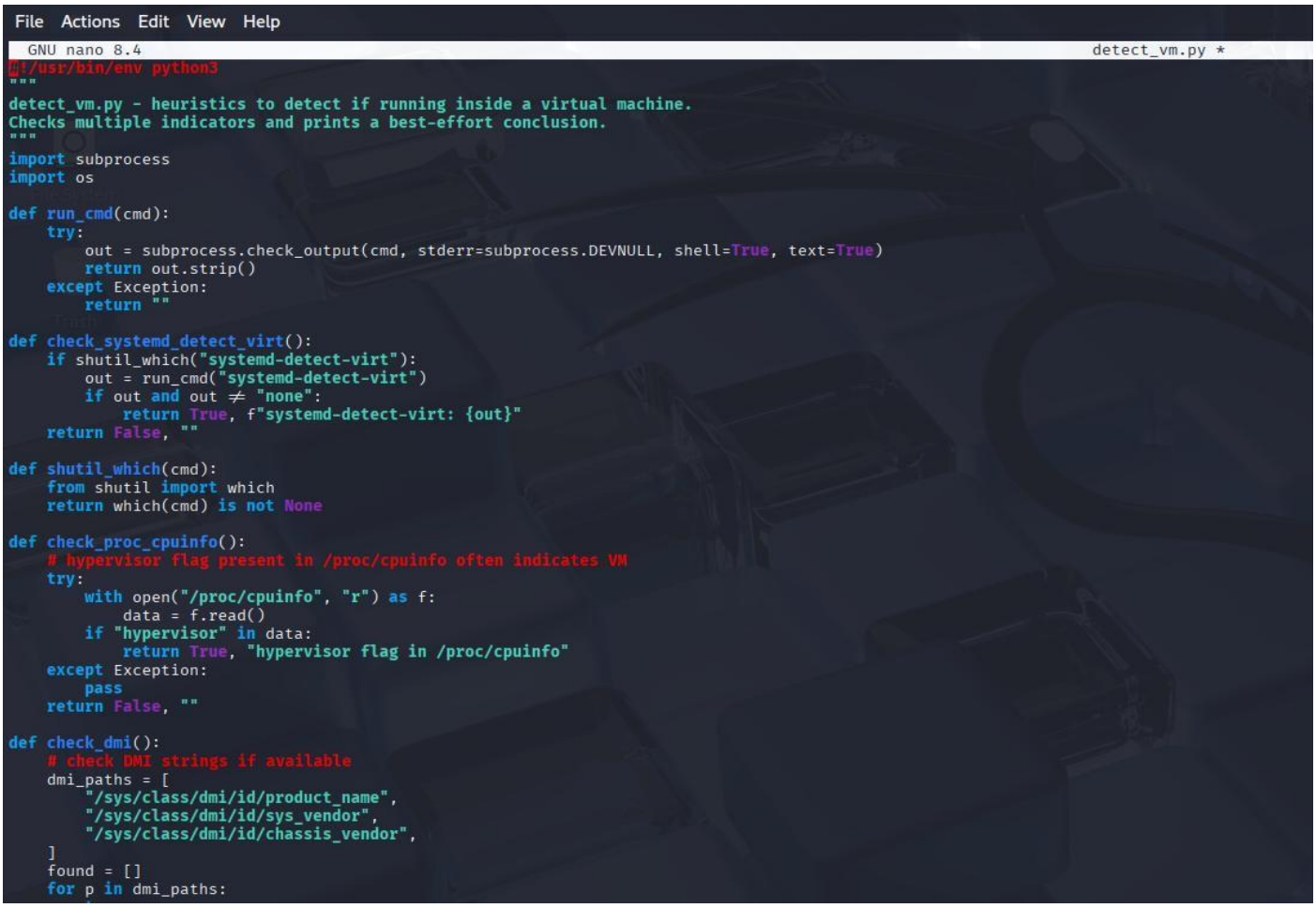


OUTPUT



- Python script to detect VM: detect_vm.py

CODE



File Actions Edit View Help

GNU nano 8.4 detect_vm.py *

```
        try:
            with open(p, "r") as f:
                v = f.read().strip().lower()
            if v:
                found.append((p, v))
        except Exception:
            continue
    # common VM signatures
    vm_signatures = ["vmware", "virtualbox", "kvm", "qemu", "microsoft corporation", "virtual", "amazon", "xen", "bochs", "parallels"]
    for path, val in found:
        for sig in vm_signatures:
            if sig in val:
                return True, f"{path} contains '{val}' (matched '{sig}')"
    return False, ""

def check_mac_oui():
    # optional: check MAC addresses for known virtual vendors
    try:
        import re, netifaces
    except Exception:
        return False, ""
    try:
        for iface in netifaces.interfaces():
            addrs = netifaces.ifaddresses(iface).get(netifaces.AF_LINK, [])
            for a in addrs:
                mac = a.get('addr')
                if mac:
                    prefix = mac.replace(":", "").upper()[:6]
                    # simple list of virtual vendor OUIs (not exhaustive)
                    virtual_ouis = {"000C29","000569","001C14","080027","525400"}  # vmware, virtualbox, etc.
                    if prefix in virtual_ouis:
                        return True, f"MAC OUI {prefix} suggests virtual NIC on {iface}"
    except Exception:
        pass
    return False, ""

def main():
    checks = []

    # 1) systemd-detect-virt (best, if available)
    if shutil_which("systemd-detect-virt"):
        out = run_cmd("systemd-detect-virt")
        if out and out != "none":
            print("Detected virtualization via systemd-detect-virt:", out)
            return

    # 2) /proc/cpuinfo hypervisor flag
    ok, reason = check_proc_cpuinfo()
    if ok:
        print("Likely running inside a VM:", reason)
        return

    # 3) DMI strings
    ok, reason = check_dmi()
    if ok:
        print("Likely running inside a VM (DMI):", reason)
        return

    # 4) optional MAC OUI check (requires netifaces)
    ok, reason = check_mac_oui()
    if ok:
        print("Likely running inside a VM (NIC OUI):", reason)
        return

    print("No strong indicators of virtualization found (host might be physical).")
    print("Note: detection is heuristic and may be inconclusive.")

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

OUTPUT

```
┌──(lakshita㉿kali)-[~/linux_process_simulation/Assignment4_BatchProcessing]
└─$ python3 detect_vm.py
Detected virtualization via systemd-detect-virt: oracle
```