

# ASSIGNMENT 4

Name: Aman Singhal

Roll No: 2301410004

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 8.0
import subprocess
import os

def batch_process(files):
    print("Starting Batch Processing ... \n")

    for file in files:
        if os.path.exists(file):
            print(f"Executing: {file}")
            result = subprocess.run(["python3", file])
            print(f"Finished: {file}\n")
        else:
            print(f"File not found: {file}")

    print("Batch Processing Complete.")

if __name__ == "__main__":
    # Add the Python files you want to execute in sequence
    files_to_run = [
        "task1.py",
        "task2.py",
        "task3.py"
    ]

    batch_process(files_to_run)
```

### OUTPUT



File Actions Edit View Help

(kali@kali)-[~]

\$ nano batch\_processor.py

(kali@kali)-[~]

\$ python3 batch\_processor.py

Starting Batch Processing ...

File not found: task1.py

File not found: task2.py

File not found: task3.py

Batch Processing Complete.

## 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

```
GNU nano 8.0
File Actions Edit View Help

import multiprocessing
import logging

logging.basicConfig(
    filename="system_startup.log",
    level=logging.INFO,
    format="%(asctime)s - %(processName)s - %(message)s"
)

def service_task(service_name):
    logging.info(f"{service_name} STARTED")
    time.sleep(2) # Simulate work
    logging.info(f"{service_name} STOPPED")

if __name__ == "__main__":
    print("Starting system boot ...")

    services = ["Network Manager", "Disk Manager", "User Authenticator"]

    processes = []

    for service in services:
        p = multiprocessing.Process(target=service_task, args=(service,))
        processes.append(p)
        p.start()

    for p in processes:
        p.join()

    print("System startup complete. Check system_startup.log for details.")
```

## OUTPUT

```
(kali㉿kali)-[~]  
$ nano system_startup.py  
  
(kali㉿kali)-[~]  
$ python3 system_startup.py  
Starting system boot ...  
System startup complete. Check system_startup.log for details.  
  
(kali㉿kali)-[~]  
$ cat system_startup.log  
2025-11-22 04:50:57,691 - Process-1 - Network Manager STARTED  
2025-11-22 04:50:57,699 - Process-3 - User Authenticator STARTED  
2025-11-22 04:50:57,698 - Process-2 - Disk Manager STARTED  
2025-11-22 04:56:09,179 - Process-1 - Network Manager STARTED  
2025-11-22 04:56:09,183 - Process-2 - Disk Manager STARTED  
2025-11-22 04:56:09,185 - Process-3 - User Authenticator STARTED  
2025-11-22 04:56:11,181 - Process-1 - Network Manager STOPPED  
2025-11-22 04:56:11,184 - Process-2 - Disk Manager STOPPED  
2025-11-22 04:56:11,187 - Process-3 - User Authenticator STOPPED
```

### 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

```
File Actions Edit View Help
GNU nano 8.0
import os

def main():
    r, w = os.pipe()

    pid = os.fork()

    if pid > 0:
        os.close(r)
        message = b"Hello from Parent Process!"
        os.write(w, message)
        os.close(w)
        os.wait()
        print("Parent: Child process finished.")
    else:
        os.close(w)
        data = os.read(r, 1024)
        os.close(r)
        print(f"Child received: {data.decode()}")
        print("Child now executing ls -l using exec()")
        os.execvp("ls", ["ls", "-l"])

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

## OUTPUT

```
(kali㉿kali)-[~]  
$ nano ipc_pipe_fork.py  
  
(kali㉿kali)-[~]  
$ python3 ipc_pipe_fork.py  
Child received: Hello from Parent Process!  
Child now executing ls -l using exec()  
total 96  
drwxrwxr-x 2 kali kali 4096 Nov 20 08:15 assignment_2  
-rw-rw-r-- 1 kali kali 598 Nov 22 04:40 batch_processor.py  
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  
-rw-rw-r-- 1 kali kali 1150 Nov 20 09:00 fcfs.py  
-rw-rw-r-- 1 kali kali 531 Nov 22 05:02 ipc_pipe_fork.py  
drwxr-xr-x 2 kali kali 4096 Aug 21 2024 Music  
drwxrwxr-x 3 kali kali 4096 Nov 20 08:24 os_simulation  
drwxr-xr-x 2 kali kali 4096 Nov 20 09:00 Pictures  
-rw-rw-r-- 1 kali kali 198 Nov 20 08:21 'process_task.py '  
drwxr-xr-x 2 kali kali 4096 Aug 21 2024 Public  
-rw-rw-r-- 1 kali kali 1739 Nov 20 09:24 round_robin.py  
-rw-rw-r-- 1 kali kali 1293 Nov 20 09:06 sjf.py  
-rw-rw-r-- 1 kali kali 558 Nov 22 04:56 system_startup.log  
-rw-rw-r-- 1 kali kali 774 Nov 22 04:56 system_startup.py  
-rw-rw-r-- 1 kali kali 763 Oct 6 00:15 task1_fork.py  
-rw-rw-r-- 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  
-rw-rw-r-- 1 kali kali 1392 Oct 6 00:27 task3_zombie_orphan.py  
-rw-rw-r-- 1 kali kali 1767 Oct 6 00:32 task4_proc_inspect.py  
-rw-rw-r-- 1 kali kali 1456 Oct 6 00:41 task5_priority.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 process finished.
```

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

## CODE

```
GNU nano 8.0
File Actions Edit View Help
import os
import sys

def main():
    r, w = os.pipe()

    pid = os.fork()

    if pid == 0:
        os.close(w)
        os.dup2(r, 0)
        os.close(r)
        os.execvp("grep", ["grep", "hello"])
    else:
        os.close(r)
        message = b"hello world from parent\nthis line does not match\nanother hello line\n"
        os.write(w, message)
        os.close(w)
        os.wait()
        print("Parent: Child finished grep execution.")

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

## OUTPUT

```
(kali㉿kali)-[~]
$ nano exec_with_pipe.py

(kali㉿kali)-[~]
$ python3 exec_with_pipe.py
hello world from parent
another hello line
Parent: Child finished grep execution.
```

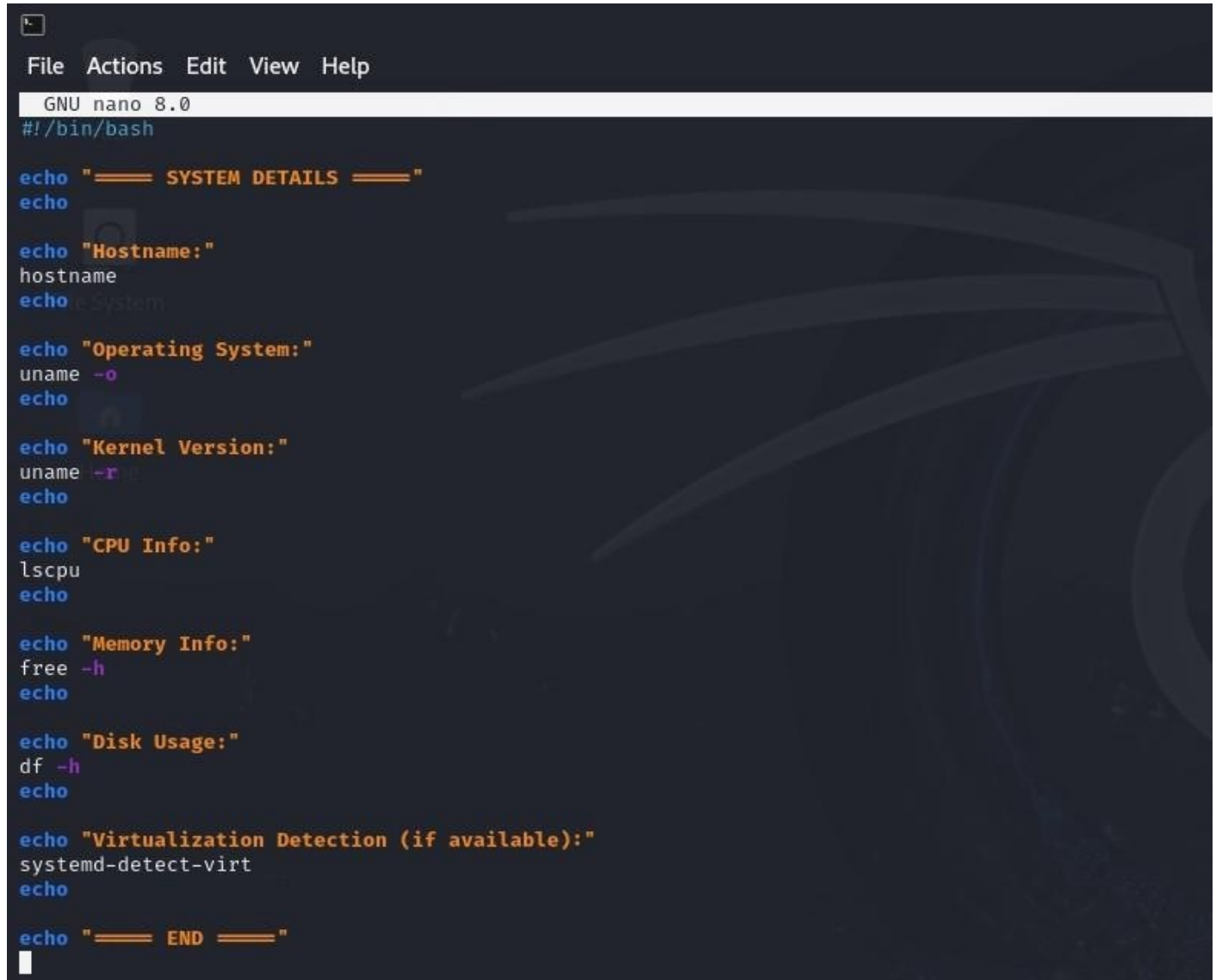


## Task 4: VM Detection and Shell Interaction

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

A screenshot of a terminal window with a dark background. At the top, there's a menu bar with 'File', 'Actions', 'Edit', 'View', and 'Help'. Below it, a status bar shows 'GNU nano 8.0' and the file path '#!/bin/bash'. The main area contains a shell script with various system information commands. The script starts with a title '==== SYSTEM DETAILS ====' and ends with '==== END ===='. It prints the hostname, operating system, kernel version, CPU info, memory info, disk usage, and a virtualization detection result using 'systemd-detect-virt'.

```
File Actions Edit View Help
GNU nano 8.0
#!/bin/bash

echo "==== SYSTEM DETAILS ===="
echo
echo "Hostname:"
hostname
echo "Operating System:"
uname -o
echo
echo "Kernel Version:"
uname -r
echo
echo "CPU Info:"
lscpu
echo
echo "Memory Info:"
free -h
echo
echo "Disk Usage:"
df -h
echo
echo "Virtualization Detection (if available):"
systemd-detect-virt
echo
echo "==== END ===="
```



## OUTPUT

```
(kali@kali)-[~]
└─$ nano system_details.sh
(kali@kali)-[~]
└─$ chmod +x system_details.sh
(kali@kali)-[~]
└─$ ./system_details.sh
===== SYSTEM DETAILS =====

Hostname:
kali

Operating System:
GNU/Linux

Kernel Version:
6.6.19-amd64

CPU Info:
Architecture:      x86_64
CPU op-mode(s):    32-bit, 64-bit
Address sizes:     40 bits physical, 48 bits virtual
Byte Order:        Little Endian
CPU(s):            4
On-line CPU(s) list: 0-3
Vendor ID:         GenuineIntel
Model name:        12th Gen Intel(R) Core(TM) i7-1255U
CPU family:        6
Model:             154
Thread(s) per core: 1
Core(s) per socket: 2
Socket(s):         2
Stepping:          4
BogoMIPS:          5222.40
Flags:             fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx rdtscp lm constant_tsc arch_perfmon nopl tsc_reliable nonstop_tsc cpuid
                    tsc_known_freq pni pclmulqdq ssse3 fma cx16 sse4_1 sse4_2 movbe popcnt aes xsave avx hypervisor lahf_lm 3dnowprefetch pti arat

Virtualization features:
Hypervisor vendor: VMware
Virtualization type: full

Caches (sum of all):
L1d:              192 KiB (4 instances)
L1i:              128 KiB (4 instances)
L2:               5 MiB (4 instances)
L3:              24 MiB (2 instances)
```

```
NUMA:
NUMA node(s):      1
NUMA node0 CPU(s): 0-3

Vulnerabilities:
Gather data sampling: Not affected
Itlb multihit:      KVM: Mitigation: VMX unsupported
L1tf:               Mitigation; PTE Inversion
Mds:               Vulnerable: Clear CPU buffers attempted, no microcode; SMT Host state unknown
Meltdown:          Mitigation; PTI
Mmio stale data:    Not affected
Retbleed:           Not affected
Spec rstack overflow: Not affected
Spec store bypass: Vulnerable
Spectre v1:         Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Spectre v2:         Mitigation; Retpolines, STIBP disabled, RSB filling, PBRSE-eIBRS Not affected
Srbds:              Not affected
Tsx async abort:    Not affected

Memory Info:
total      used      free      shared  buff/cache   available
Mem:      1.9Gi      817Mi      829Mi       7.6Mi       471Mi       1.1Gi
Swap:      1.0Gi          0B        1.0Gi

Disk Usage:
Filesystem      Size  Used Avail Use% Mounted on
udev            943M   0    943M   0% /dev
tmpfs           197M  1.3M  196M   1% /run
/dev/sda1        79G   22G   54G  29% /
tmpfs           984M   0    984M   0% /dev/shm
tmpfs           5.0M   0    5.0M   0% /run/lock
tmpfs           197M  128K  197M   1% /run/user/1000

Virtualization Detection (if available):
vmware

===== END =====
```

- Python script to detect VM: detect\_vm.py

## CODE

```
kali@kali: ~
File Actions Edit View Help
GNU nano 8.0 detect_vm.py *
import subprocess

def check_systemd_virt():
    try:
        output = subprocess.check_output(["systemd-detect-virt"], text=True).strip()
        return output
    except:
        return "unknown"

def check_dmi_field(field):
    try:
        with open(f"/sys/class/dmi/id/{field}", "r") as f:
            return f.read().strip()
    except:
        return "unknown"

def check_cpu_flags():
    try:
        with open("/proc/cpuinfo", "r") as f:
            data = f.read()
            return "hypervisor" in data
    except:
        return False

def detect_vm():
    print("==== VM Detection ==== \n")

    systemd = check_systemd_virt()
    print("systemd-detect-virt:", systemd)

    bios_vendor = check_dmi_field("bios_vendor")
    product = check_dmi_field("product_name")
    print("BIOS Vendor:", bios_vendor)
    print("Product Name:", product)

    hypervisor_flag = check_cpu_flags()
    print("CPU Hypervisor Flag:", hypervisor_flag)

    vm_keywords = ["vmware", "virtualbox", "qemu", "kvm", "hyper-v", "xen"]
    combined = (bios_vendor + " " + product).lower()

    detected = "Yes" if any(k in combined for k in vm_keywords) or hypervisor_flag else "No"
    print("\nRunning Inside VM:", detected)

    print("\n==== END ====")

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

## OUTPUT

```
(kali@kali)-[~]
$ nano detect_vm.py

(kali@kali)-[~]
$ python3 detect_vm.py
==== VM Detection ====

systemd-detect-virt: vmware
BIOS Vendor: Phoenix Technologies LTD
Product Name: VMware Virtual Platform
CPU Hypervisor Flag: True

Running Inside VM: Yes

==== END =====
```