

ASSIGNMENT 4

Name: Payal Rathore

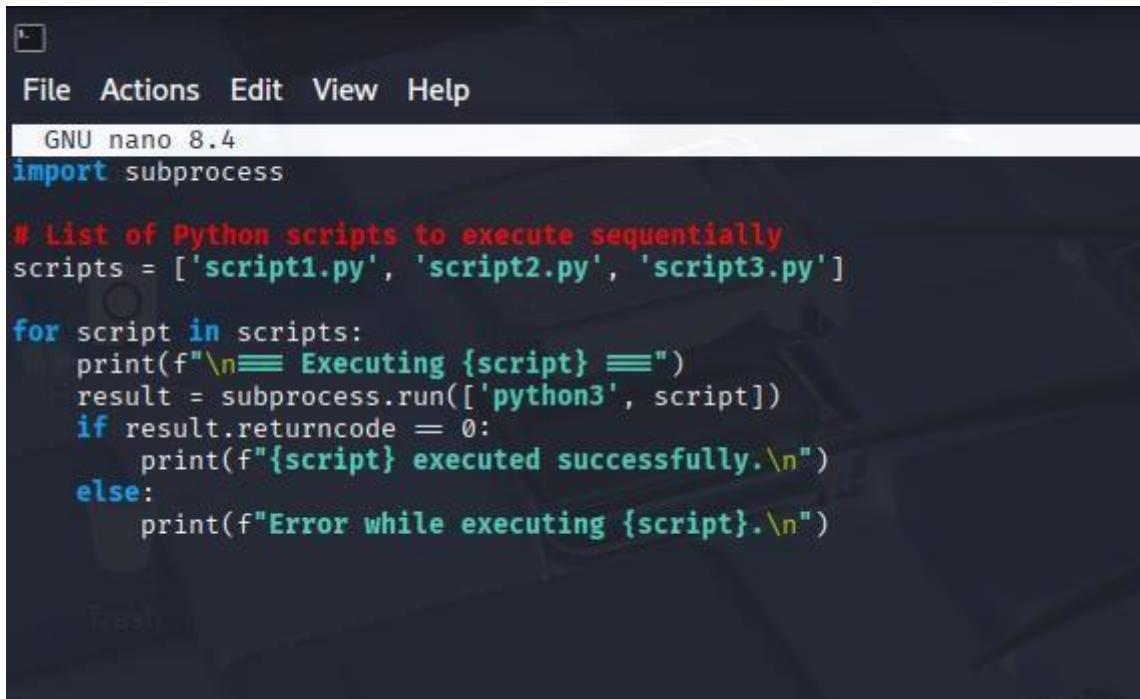
Roll No: 2301410022

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

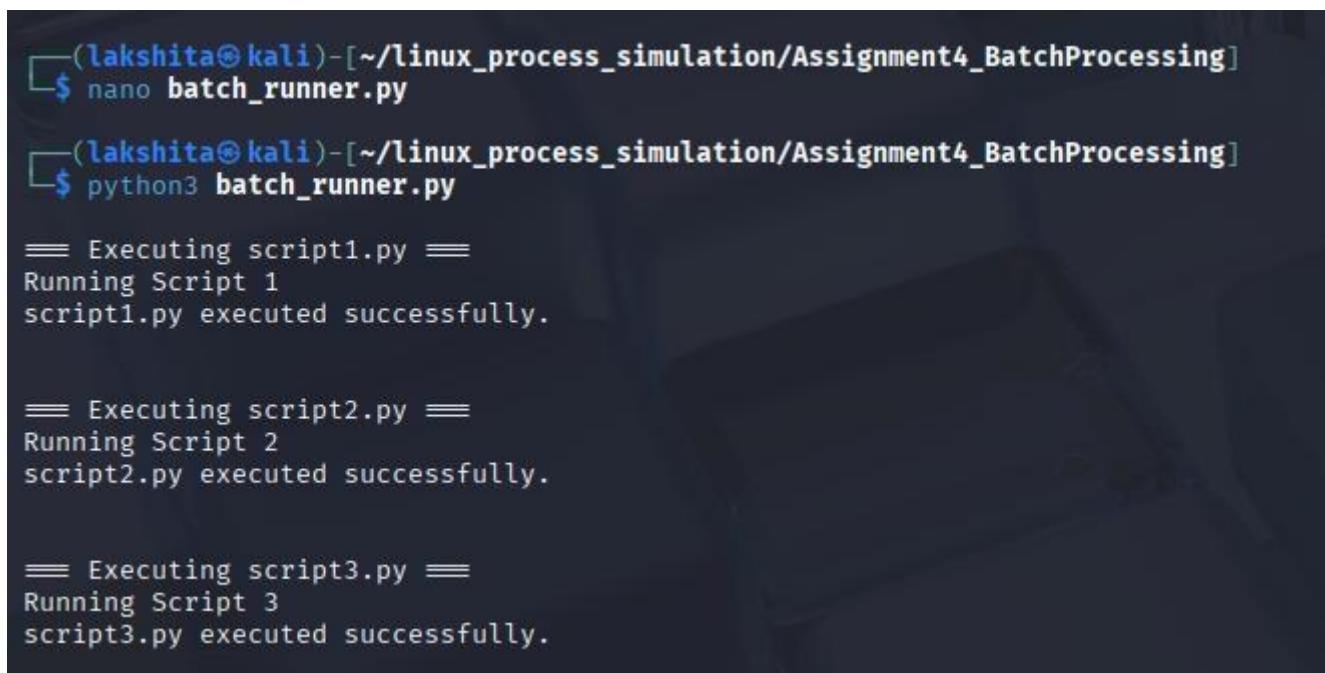


```
File Actions Edit View Help
GNU nano 8.4
import subprocess

# 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_simulation
```

```
File Actions Edit View Help
GNU nano 8.4
/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

```
lakshita@kali: ~/linux_process_simulation$ nano ipc_pipe_fork.py
GNU nano 8.4
#!/usr/bin/env python3
"""
ipc_pipe_fork.py
Simple IPC using os.pipe() and os.fork():
Parent sends a message to child using the pipe, child reads it and responds.
"""

import os
import sys

def parent_child_communication():
    # create a pipe: r, w are file descriptors
    r, w = os.pipe()

    pid = os.fork()
    if pid == 0:
        # child process
        os.close(w) # close write end in child
        rfd = os.fdopen(r, 'r')
        msg = rfd.read() # read everything the parent writes
        print(f"Child (pid {os.getpid()}): received from parent: {msg.strip()}")
        rfd.close()
        sys.exit(0)
    else:
        # parent process
        os.close(r) # close read end in parent
        wfd = os.fdopen(w, 'w')
        message = "Hello child! This is parent.\n"
        wfd.write(message)
        wfd.flush()
        wfd.close()
        # wait for child to finish
        pid_done, status = os.waitpid(pid, 0)
        print(f"Parent: child {pid_done} exited with status {status}")

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

OUTPUT

```
(lakshita㉿kali)-[~/linux_process_simulation/Assignment4_BatchProcessing]
$ python3 ipc_pipe_fork.py

Child (pid 31786): received from parent: Hello child! This is parent.
Parent: child 31786 exited with status 0
```

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

CODE

```

lakshita@kali: ~/linux_process_simulation/Assignment4_BatchProcessing] exec_v
File Actions Edit View Help
GNU nano 8.4
#!/usr/bin/env python3
"""

exec_with_pipe.py
Parent writes lines to a pipe, children exec 'grep' to filter lines.
Demonstrates fork + exec + pipes.

"""

import os
import sys
import time

def run_exec_pipe():
    r, w = os.pipe()
    pid = os.fork()
    if pid == 0:
        # Child: execute grep to filter lines containing "ok"
        os.dup2(r, 0)    # replace stdin with read-end of pipe
        os.close(w)
        os.close(r)
        # exec grep
        os.execvp("grep", ["grep", "ok"])
        # if exec fails:
        print("Exec failed", file=sys.stderr)
        sys.exit(1)
    else:
        # Parent: write messages to pipe, close and wait
        os.close(r)
        wfd = os.fdopen(w, 'w')
        lines = ["this is ok\n", "this is not\n", "ok indeed\n"]
        for L in lines:
            wfd.write(L)
            wfd.flush()
            time.sleep(0.2)
        wfd.close()
        # wait for child to finish; capture exit status
        pid_done, status = os.waitpid(pid, 0)
        exit_code = os.WEXITSTATUS(status)
        print(f"Parent: child {pid_done} exited with code {exit_code}")

if __name__ == "__main__":
    run_exec_pipe()

```

OUTPUT

```

[(lakshita@kali)-[~/linux_process_simulation/Assignment4_BatchProcessing]
$ python3 exec_with_pipe.py

this is ok
ok indeed
Parent: child 33442 exited with code 0

```

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

```
lakshita@kali: ~/linux_process_simulation/Assignment4_
File Actions Edit View Help
GNU nano 8.4
#!/usr/bin/env bash
# system_info.sh - print system details (some commands may need sudo)

echo "==== uname -a ===="
uname -a
echo

echo "==== lscpu ===="
lscpu
echo

echo "==== free -h ===="
free -h
echo

echo "==== lsblk ===="
lsblk
echo

echo "==== ip addr (show interfaces) ===="
ip -c addr
echo

echo "==== last reboot (uptime) ===="
uptime
echo

# dmidecode may require root privileges; print a short note if not accessible
if command -v dmidecode >/dev/null 2>&1; then
    echo "==== dmidecode -t system (requires sudo) ===="
    if [ "$(id -u)" -eq 0 ]; then
        dmidecode -t system
    else
        echo "dmidecode available but requires sudo. Run: sudo dmidecode -t system"
    fi
else
    echo "dmidecode not installed or not available."
fi

echo
echo "==== lspci (if available) ===="
if command -v lspci >/dev/null 2>&1; then
    lspci | head -n 20
else
    echo "lspci not available"
fi
```

OUTPUT

```
lakshita@kali:~/linux_process_simulation/Assignment4_BatchProcessing]
$ ./system_info.sh

==== uname -a ====
Linux kali 6.12.33+kali-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.12.33-1kalii (2025-06-25) x86_64 GNU/Linux

==== lscpu ====
Architecture:          x86_64
CPU op-mode(s):       32-bit, 64-bit
Address sizes:        39 bits physical, 48 bits virtual
Byte Order:           Little Endian
CPU(s):               1
On-line CPU(s) list: 0
Vendor ID:            GenuineIntel
Model name:           11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
CPU family:           6
Model:                140
Thread(s) per core:   1
Core(s) per socket:   1
Socket(s):            1
Stepping:             1
BogoMIPS:             5606.39
Flags:                fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_good nopl xttopology ssse3 fma cx16 pcid sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch ibrs_enhanced fsgsbbase bmi1 avx2 bmi2 invpcid rd arch_capabilities

Virtualization features:
Hypervisor vendor:   KVM
Virtualization type: full
Caches (sum of all):
L1d:                 48 KiB (1 instance)
L1i:                 32 KiB (1 instance)
L2:                  1.3 MiB (1 instance)
L3:                  12 MiB (1 instance)
NUMA:
NUMA node(s):         1
NUMA node0 CPU(s):    0
Vulnerabilities:
Gather data sampling: Not affected
Indirect target selection: Mitigation; Aligned branch/return thunks
Itlb multihit:        Not affected
L1tf:                 Not affected
Mds:                  Not affected
Meltdown:             Not affected
Mmio stale data:     Not affected
Reg file data sampling: Not affected
Retbleed:              Mitigation; Enhanced IBRS
Spec rstack overflow: Not affected
Spec store bypass:    Vulnerable
Spectre v1:             Mitigation; usercopy/swaps barriers and __user pointer sanitization
Spectre v2:             Mitigation; Enhanced / Automatic IBRS; PBRSB-eIBRS SW sequence; BHI SW loop, KVM SW loop
Srbds:                 Not affected
```

```

lakshita@kali:~/linux_process_simulation/Assignment4_BatchProcessing
File Actions Edit View Help
Srbds: Not affected
Tx sync abort: Not affected
== free -h ==
Mem: 1.96Gi 736Mi 892Mi 17Mi 502Mi 1.2Gi
Swap: 1.3Gi 0B 1.3Gi

== lsblk ==
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
sda 8:0 0 25G 0 disk
└─sda1 8:1 0 23.7G 0 part /
└─sda2 8:2 0 1K 0 part
└─sda5 8:5 0 1.3G 0 part [SWAP]
sr0 11:0 1 1024M 0 rom

== ip addr (show interfaces) ==
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host noprefixroute
            valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:b2:1c:9f brd ff:ff:ff:ff:ff:ff
        inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute eth0
            valid_lft 68672sec preferred_lft 68672sec
        inet6 fd17:625c:f037:2:a0e1:60d5:d9f4:e0b4/64 scope global temporary dynamic
            valid_lft 86268sec preferred_lft 14268sec
        inet6 fd17:625c:f037:2:a0e0:27ff:feb2:1c9f/64 scope global dynamic mngtmpaddr noprefixroute
            valid_lft 86268sec preferred_lft 14268sec
        inet6 fe80::a0e0:27ff:feb2:1c9f/64 scope link noprefixroute
            valid_lft forever preferred_lft forever

== last reboot (uptime) ==
19:29:18 up 4:55, 1 user, load average: 0.06, 0.11, 0.07

== dmidecode -t system (requires sudo) ==
dmidecode available but requires sudo. Run: sudo dmidecode -t system

== lspci (if available) ==
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)
00:02.0 VGA compatible controller: VMware SVGA II Adapter
00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet Controller (rev 02)
00:04.0 System peripheral: Innotek Systemberatung GmbH VirtualBox Guest Service
00:05.0 Multimedia audio controller: Intel Corporation 82801AA AC'97 Audio Controller (rev 01)
00:06.0 USB controller: Apple Inc. KeyLargo/Intrepid USB
00:07.0 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
00:08.0 USB controller: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) USB2 EHCI Controller
00:0d.0 SATA controller: Intel Corporation 82801HM/HEM (ICH8M/ICH8M-E) SATA Controller [AHCI mode] (rev 02)

```

- Python script to detect VM: detect_vm.py

CODE

```

File Actions Edit View Help
GNU nano 8.4
#!/usr/bin/env python3
"""
detect_vm.py - heuristics to detect if running inside a virtual machine.
Checks multiple indicators and prints a best-effort conclusion.
"""

import subprocess
import os

def run_cmd(cmd):
    try:
        out = subprocess.check_output(cmd, stderr=subprocess.DEVNULL, shell=True, text=True)
        return out.strip()
    except Exception:
        return ""

def check_systemd_detect_virt():
    if shutil.which("systemd-detect-virt"):
        out = run_cmd("systemd-detect-virt")
        if out and out != "none":
            return True, f"systemd-detect-virt: {out}"
        return False, ""
    return False, ""

def shutil_which(cmd):
    from shutil import which
    return which(cmd) is not None

def check_proc_cpuinfo():
    # hypervisor flag present in /proc/cpuinfo often indicates VM
    try:
        with open("/proc/cpuinfo", "r") as f:
            data = f.read()
            if "hypervisor" in data:
                return True, "hypervisor flag in /proc/cpuinfo"
    except Exception:
        pass
    return False, ""

def check_dmi():
    # check DMI strings if available
    dmi_paths = [
        "/sys/class/dmi/id/product_name",
        "/sys/class/dmi/id/sys_vendor",
        "/sys/class/dmi/id/chassis_vendor",
    ]
    found = []
    for p in dmi_paths:
        if os.path.exists(p):
            found.append(p)
    if found:
        return True, f"DMI strings found: {found}"
    return False, ""

```

```

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_cpufifo()
    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

```