ASSIGNMENT 4

NAME- Tanmay Gaur
ROLL NO.- 2301010419
COURSE- BTech CSE Core

Task 1: Batch Processing Simulation (Python)

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

CODE:

```
  GNU nano 8.4                                                    task1.py
import subprocess
import sys

scripts = ['script1.py', 'script2.py', 'script3.py']

if __name__ == "__main__":
    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} (exit code {result.returncode}).\n")
            sys.exit(result.returncode)
```

OUTPUT:

```
┌──(anjali㉿Anjali)-[~]
└─$ nano task1.py

┌──(anjali㉿Anjali)-[~]
└─$ python3 task1.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:

```
  GNU nano 8.4                                                    task2.py *
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())

def worker(name, duration):
    logging.info(f"Worker '{name}' starting (PID={os.getpid()})")
    t0 = time.time()
    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("System startup simulation beginning")

    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)

        time.sleep(0.2)

    for p in procs:
        p.join()

    logging.info("Startup simulation complete")

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

OUTPUT:

```
┌──(anjali㉿Anjali)-[~]
└─$ nano task2.py

┌──(anjali㉿Anjali)-[~]
└─$ python3 task2.py
System startup simulation beginning
Worker 'init_services' starting (PID=328)
Worker 'mount_filesystems' starting (PID=329)
Worker 'network_manager' starting (PID=330)
Worker 'login_manager' starting (PID=331)
Worker 'cron_jobs' starting (PID=332)
Worker 'mount_filesystems' finished (PID=329) elapsed=1.00s
Worker 'cron_jobs' finished (PID=332) elapsed=0.80s
Worker 'init_services' finished (PID=328) elapsed=2.00s
Worker 'login_manager' finished (PID=331) elapsed=1.50s
Worker 'network_manager' finished (PID=330) elapsed=3.00s
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:

```
 GNU nano 8.4                                                    task3_ipc.py
import os
import sys
# ipc_pipe_fork.py
# Simple IPC using os.pipe() and os.fork()
# Parent sends a message to child; child reads it and prints it.

def parent_child_communication():
    # Create pipe (r = read end, w = write end)
    r, w = os.pipe()

    pid = os.fork()

    if pid == 0:
        # -------- CHILD PROCESS --------
        os.close(w)  # Child doesn't write

        rfd = os.fdopen(r, 'r')
        msg = rfd.read()  # read everything sent by parent
        print(f"Child (pid {os.getpid()}): received from parent: {msg.strip()}")

        rfd.close()
        sys.exit(0)

    else:
        # -------- PARENT PROCESS --------
        os.close(r)  # Parent doesn't read

        wfd = os.fdopen(w, 'w')
        message = "Hello child! This is the 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 :

```
┌──(anjali㉿Anjali)-[~]
└─$ nano task3_ipc.py

┌──(anjali㉿Anjali)-[~]
└─$ python3 task3_ipc.py
Child (pid 354): received from parent: Hello child! This is the parent.
Parent: child 354 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:

```
  GNU nano 8.4                                                    task3_exc.py *
# Parent writes lines to a pipe, child execs 'grep' to filter "ok"
# Demonstrates: fork + pipe + dup2 + execvp

import os
import sys
import time

def run_exec_pipe():
    # Create pipe
    r, w = os.pipe()
    pid = os.fork()

    if pid == 0:
        # ---------------- CHILD PROCESS ----------------
        # Close write-end in child
        os.close(w)

        # Replace child's STDIN with the read-end of the pipe
        os.dup2(r, 0)

        # Close original file descriptor after duplicating
        os.close(r)

        # Execute: grep "ok"
        os.execvp("grep", ["grep", "ok"])

        # If exec fails:
        print("Exec failed!", file=sys.stderr)
        sys.exit(1)

    else:
        # ---------------- PARENT PROCESS ----------------
        os.close(r)  # Parent does not read

        wfd = os.fdopen(w, 'w')

        lines = [
            "this is ok\n",
            "this is not\n",
            "ok indeed\n"
        ]

        for line in lines:
            wfd.write(line)
            wfd.flush()
            time.sleep(0.2)

        wfd.close()

        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 :

```
┌──(anjali㉿Anjali)-[~]
└─$ nano task3_exc.py

┌──(anjali㉿Anjali)-[~]
└─$ python3 task3_exc.py
this is ok
ok indeed
Parent: child 362 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:

```
  GNU nano 8.4                                                    system.info.sh *
#!/usr/bin/env bash
# system_info.sh - Print detailed system information

echo "============================="
echo "     SYSTEM INFORMATION"
echo "============================="

echo
echo "--- Kernel Version ---"
uname -a
echo

echo "--- CPU Information (lscpu) ---"
lscpu || echo "lscpu command not available"
echo

echo "--- Memory Info (free -h) ---"
free -h || echo "free command not available"
echo

echo "--- Block Devices (lsblk) ---"
lsblk || echo "lsblk command not available"
echo

echo "--- Network Interfaces (ip addr) ---"
ip -c addr || echo "ip command not available"
echo

echo "--- System Uptime ---"
uptime || echo "uptime command not available"
echo

echo "--- DMI / Hardware Vendor Info ---"
if command -v dmidecode >/dev/null 2>&1; then
    if [ "$(id -u)" -eq 0 ]; then
        dmidecode -t system
    else
        echo "dmidecode exists but requires sudo permissions."
        echo "Run: sudo dmidecode -t system"
    fi
else
    echo "dmidecode not installed."
fi

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

echo
echo "======= END OF REPORT ======="
```

OUTPUT :

```
┌──(anjali㉿Anjali)-[~]
└─$ ./system.info.sh
============================
     SYSTEM INFORMATION
============================

--- Kernel Version ---
Linux Anjali 6.6.87.2-microsoft-standard-WSL2 #1 SMP PREEMPT_DYNAMIC Thu Jun  5 18:30:46 UTC 2025 x86_64 GNU/Linux

--- CPU Information (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):                   4
  On-line CPU(s) list:    0-3
Vendor ID:                GenuineIntel
  Model name:             11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz
    CPU family:           6
    Model:                140
    Thread(s) per core:   2
    Core(s) per socket:   2
    Socket(s):            1
    Stepping:             2
    BogoMIPS:             5990.40
    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 pdpe1gb rdtscp lm constant_tsc arch_perfmon rep
                          _good nopl xtopology tsc_reliable nonstop_tsc cpuid tsc_known_freq pni pclmulqdq vmx ssse3 fma cx16 pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer
                           aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep b
                          mi2 erms invpcid avx512f avx512dq rdseed adx smap avx512ifma clflushopt clwb avx512cd sha_ni avx512bw avx512vl xsaveopt xsavec xgetbv1 xsaves vnmi avx512vbmi umip
                           avx512_vbmi2 gfni vaes vpclmulqdq avx512_vnni avx512_bitalg avx512_vpopcntdq rdpid movdiri movdir64b fsrm avx512_vp2intersect md_clear flush_l1d arch_capabilitie
                          s
Virtualization features:
  Virtualization:         VT-x
  Hypervisor vendor:      Microsoft
  Virtualization type:    full
Caches (sum of all):
  L1d:                    96 KiB (2 instances)
  L1i:                    64 KiB (2 instances)
  L2:                     2.5 MiB (2 instances)
  L3:                     6 MiB (1 instance)
NUMA:
  NUMA node(s):           1
  NUMA node0 CPU(s):      0-3
Vulnerabilities:
  Gather data sampling:   Not affected
  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:      Mitigation; Speculative Store Bypass disabled via prctl
  Spectre v1:             Mitigation; usercopy/swapgs barriers and __user pointer sanitization
  Spectre v2:             Mitigation; Enhanced / Automatic IBRS; IBPB conditional; RSB filling; PBRSB-eIBRS SW sequence; BHI SW loop, KVM SW loop
  Srbds:                  Not affected
  Tsx async abort:        Not affected

--- Memory Info (free -h) ---
               total        used        free      shared  buff/cache   available
Mem:           3.7Gi       364Mi       3.2Gi       3.4Mi       132Mi       3.3Gi
Swap:          1.0Gi          0B       1.0Gi

--- Block Devices (lsblk) ---
NAME MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda    8:0    0 388.4M  1 disk
sdb    8:16   0   186M  1 disk
sdc    8:32   0     1G  0 disk [SWAP]
sdd    8:48   0     1T  0 disk /mnt/wslg/distro
                               /

--- Network Interfaces (ip addr) ---
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
    inet 10.255.255.254/32 brd 10.255.255.254 scope global lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host proto kernel_lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1428 qdisc mq state UP group default qlen 1000
    link/ether 00:15:5d:6b:12:17 brd ff:ff:ff:ff:ff:ff
    inet 172.19.177.147/20 brd 172.19.191.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::215:5dff:fe6b:1217/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever

--- System Uptime ---
 22:45:35 up 51 min,  1 user,  load average: 0.00, 0.00, 0.00

--- DMI / Hardware Vendor Info ---
dmidecode exists but requires sudo permissions.
Run: sudo dmidecode -t system

--- PCI Devices (lspci) ---
lspci not available.

======= END OF REPORT =======
```

- Python script to detect VM: detect_vm.py

CODE:

```
  GNU nano 8.4                                                    task4_detect.py *
# detect_vm.py - Detect whether system is running inside a Virtual Machine

import subprocess
import os

VM_KEYWORDS = [
    "kvm", "qemu", "vmware", "virtualbox",
    "hyper-v", "xen", "bochs", "parallels",
    "google", "amazon", "azure"
]

def check_lscpu():
    """Check virtualization info from lscpu."""
    try:
        output = subprocess.check_output(["lscpu"]).decode().lower()
        for key in VM_KEYWORDS:
            if key in output:
                return True, f"Detected VM keyword '{key}' in lscpu"
        return False, "No VM indicators found in lscpu"
    except FileNotFoundError:
        return False, "lscpu not found"

def check_dmi():
    """Check DMI sys_vendor / product_name."""
    paths = [
        "/sys/class/dmi/id/product_name",
        "/sys/class/dmi/id/sys_vendor"
    ]
    collected = ""

    for p in paths:
        try:
            collected += open(p).read().lower() + " "
        except:
            pass

    for key in VM_KEYWORDS:
        if key in collected:
            return True, f"Detected VM keyword '{key}' in DMI info"

    return False, "No virtualization info found in DMI"

def check_cpuinfo():
    """Check hypervisor flag in cpuinfo."""
    try:
        data = open("/proc/cpuinfo").read().lower()
        if "hypervisor" in data:
            return True, "Found 'hypervisor' flag in cpuinfo"
        return False, "No hypervisor flag in cpuinfo"
    except:
        return False, "cpuinfo not readable"

def detect_vm():
    print("====== VM DETECTION REPORT ======\n")

    checks = [
        ("LSCPU", check_lscpu()),
        ("DMI", check_dmi()),
        ("CPUINFO", check_cpuinfo())
    ]

    vm_found = False
    for label, (status, message) in checks:
        print(f"[{label}] {message}")
        if status:
            vm_found = True

    print("\n================================")
    if vm_found:
        print("RESULT: Virtual Machine detected")
    else:
        print("RESULT: No Virtual Machine detected")
    print("================================")

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

OUTPUT:

```
┌──(anjali㉿Anjali)-[~]
└─$ nano task4_detect.py

┌──(anjali㉿Anjali)-[~]
└─$ python3 task4_detect.py
====== VM DETECTION REPORT ======

[LSCPU] Detected VM keyword 'kvm' in lscpu
[DMI] No virtualization info found in DMI
[CPUINFO] Found 'hypervisor' flag in cpuinfo

==================================
RESULT: Virtual Machine detected
==================================
```