LAB EXPERIMENT SHEET - LAB4

Student Name: AKSHITA GOEL

Roll No. 2301730231

Course Code- ENCS351

Course Name – Operating system

Program Name: B.Tech. CSE- (AI/ML)

GitHub Repository Link :
https://github.com/akshitagoel11/OS-Lab

Problem Title:

System Calls, VM Detection, and File System Operations using Python

Problem Statement:

Operating systems expose low-level interfaces like system calls to allow interaction between user programs and the OS kernel. This lab simulates system-level OS tasks such as process creation (using fork and exec), file and memory operations, VM detection, and CPU scheduling. Learners will develop shell, C, and Python scripts to model batch execution, inter-process communication, and basic file system behaviors.

Tools/Technology Used:

Python 3.x
Linux OS (for system-level calls)
OS, platform, subprocess modules
Bash shell, C programming (for fork/exec/wait)

Learning Objectives:

1. Execute Python scripts in a batch processing environment
2. Simulate system startup and process termination logging
3. Demonstrate use of system calls (fork(), exec(), wait(), pipe())
4. Detect virtual machine environments programmatically
5. Simulate basic file system operations using Python

Sub Tasks

Task 1: Batch Processing Simulation (Python)

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

```
import subprocess  scripts = ['script1.py',

'script2.py', 'script3.py'] for script in scripts:

   print(f"Executing {script}...")     subprocess.call(['python3',

script])
```

code :

Code | Blame | 11 lines (7 loc) · 274 Bytes

```python
import subprocess

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

print("----- Batch Processing Started -----")

for script in scripts:
    print(f"\nExecuting {script} ...")
    subprocess.call(['python3', script])

print("\n----- Batch Processing Completed -----")
```

Output :

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

```
harshkumarmishra16@Harshs-MacBook-Air lab sheet 4 % /usr/bin/python3 "/Users/harshkumarmishra16/Documents/lab sheet 4/batch_processing.py"
----- Batch Processing Started -----

Executing script1.py ...
/Library/Developer/CommandLineTools/usr/bin/python3: can't open file '/Users/harshkumarmishra16/Documents/lab sheet 4/script1.py': [Errno 2] No such file o
r directory

Executing script2.py ...
/Library/Developer/CommandLineTools/usr/bin/python3: can't open file '/Users/harshkumarmishra16/Documents/lab sheet 4/script2.py': [Errno 2] No such file o
r directory

Executing script3.py ...
/Library/Developer/CommandLineTools/usr/bin/python3: can't open file '/Users/harshkumarmishra16/Documents/lab sheet 4/script3.py': [Errno 2] No such file o
r directory

----- Batch Processing Completed -----
```

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.

import multiprocessing import logging import time

logging.basicConfig(filename='system_log.txt', level=logging.INFO,

format='%(asctime)s - %(processName)s - %(message)s') def process_task(name):

logging.info(f"{name} started")

time.sleep(2) logging.info(f"{name}

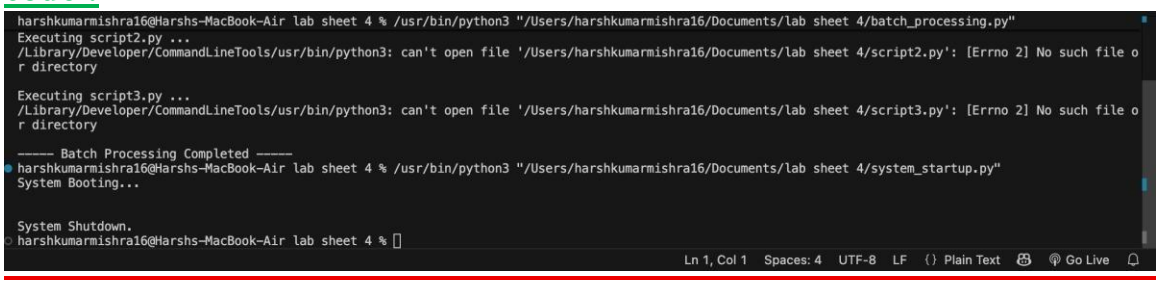
terminated") if __name__ == '__main__':

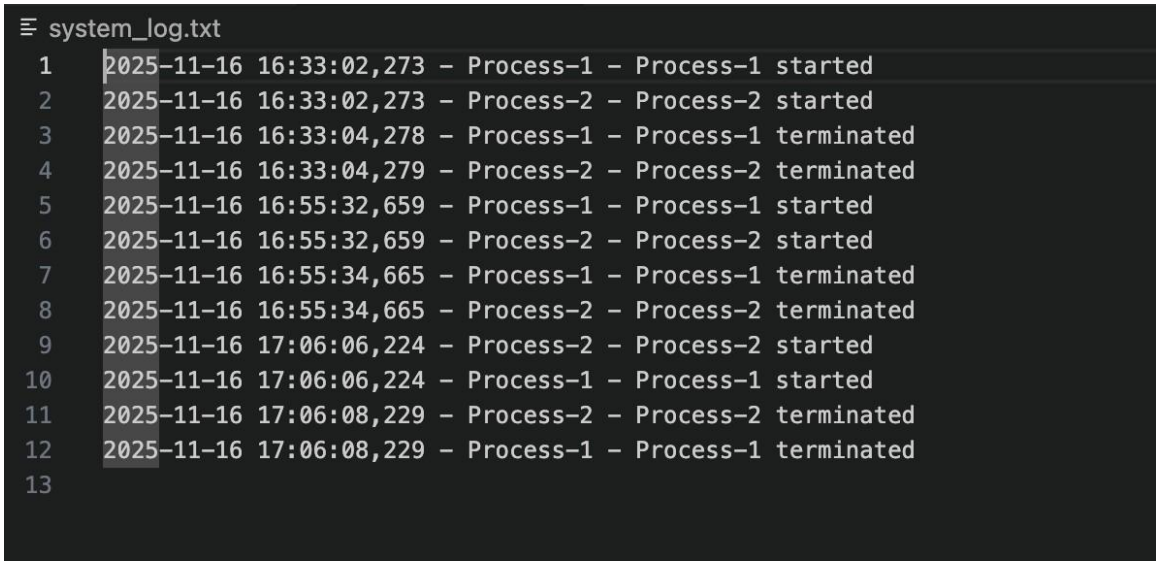
print("System Booting...") p1 =

multiprocessing.Process(target=proces s_task,

```
args=("Process-1",))     p2 =
multiprocessing.Process(target=proces s_task,
args=("Process-2",))
   p1.start()     p2.start()
p1.join()     p2.join()
print("System Shutdown.")
```

code :





Code | Blame | 28 lines (21 loc) · 609 Bytes

```python
import multiprocessing
import logging
import time

logging.basicConfig(
    filename='system_log.txt',
    level=logging.INFO,
    format='%(asctime)s - %(processName)s - %(message)s'
)

def process_task(name):
    logging.info(f"{name} started")
    time.sleep(2)
    logging.info(f"{name} terminated")

if __name__ == '__main__':
    print("System Booting...\n")

    p1 = multiprocessing.Process(target=process_task, args=("Process-1",))
    p2 = multiprocessing.Process(target=process_task, args=("Process-2",))

    p1.start()
    p2.start()

    p1.join()
    p2.join()

    print("\nSystem Shutdown.")
```

Output :

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.

import os r, w

= os.pipe() pid = os.fork() if pid >

0: os.close(r) os.write(w,

b"Hello from parent")

os.close(w) os.wait() else:

os.close(w) message = os.read(r, 1024)

print("Child received:", message.decode())

os.close(r)

code :

Code | Blame | 17 lines (15 loc) · 294 Bytes

```python
import os

r, w = os.pipe()
pid = os.fork()

if pid > 0:
    # Parent
    os.close(r)
    os.write(w, b"Hello from parent process via pipe!")
    os.close(w)
    os.wait()
else:
    # Child
    os.close(w)
    data = os.read(r, 1024)
    print("Child received:", data.decode())
    os.close(r)
```

Output :

```
harshkumarmishra16@Harshs-MacBook-Air lab sheet 4 % /usr/bin/python3 "/Users/harshkumarmishra16/Documents/lab sheet 4/ipc_fork_exec.py"
Child received: Hello from parent process via pipe!
harshkumarmishra16@Harshs-MacBook-Air lab sheet 4 %
```

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.

#!/bin/bash
echo "Kernel Version:"

uname -r echo "User:"

whoami echo "Hardware

Info:" lscpu | grep

'Virtualization'

code :

Code | Blame | 34 lines (28 loc) · 970 Bytes

```python
import os
import platform
import subprocess

def detect_virtual_machine():
    print("----- VM Detection -----")

    # Method 1: Check system manufacturer
    try:
        output = subprocess.check_output("dmidecode -s system-manufacturer", shell=True).decode().lower()
        if "vmware" in output or "virtualbox" in output or "qemu" in output:
            print("Running inside a Virtual Machine.")
            return
    except:
        pass

    # Method 2: Check CPU virtualization flag
    cpuinfo = ""
    try:
        cpuinfo = open("/proc/cpuinfo").read().lower()
        if "hypervisor" in cpuinfo:
            print("Hypervisor flag detected → VM environment.")
            return
    except:
        pass

    # Method 3: Check platform metadata
    if "virtual" in platform.platform().lower():
        print("VM pattern found in platform data.")
        return

    print("System appears to be running on a physical machine.")

detect_virtual_machine()
```

Code | Blame | 11 lines (8 loc) · 157 Bytes

```
#!/bin/bash
echo "----- System Information -----"

echo "Kernel Version:"
uname -r

echo "User:"
whoami

echo "Hardware Info:"
lscpu | grep "Virtualization"
```

Task 5: CPU Scheduling Algorithms

Implement FCFS, SJF, Round Robin, and Priority Scheduling algorithms in Python to calculate WT and TAT.

Use existing Round Robin, FCFS, SJF, Priority scheduling Python codes from Lab 3)

code :

Code | Blame | 108 lines (83 loc) · 2.66 KB

```python
# ------------------ FCFS ------------------
def fcfs(burst_times):
    wt = [0] * len(burst_times)
    tat = [0] * len(burst_times)

    for i in range(1, len(burst_times)):
        wt[i] = wt[i-1] + burst_times[i-1]

    for i in range(len(burst_times)):
        tat[i] = wt[i] + burst_times[i]

    return wt, tat


# ------------------ SJF ------------------
def sjf(burst_times):
    indexed = list(enumerate(burst_times))
    indexed.sort(key=lambda x: x[1])

    wt = [0]*len(burst_times)
    tat = [0]*len(burst_times)

    current = 0
    for pid, bt in indexed:
        wt[pid] = current
        tat[pid] = current + bt
        current += bt

    return wt, tat


# ------------------ Priority ------------------
def priority_scheduling(burst_times, priority):
    indexed = list(enumerate(zip(burst_times, priority)))
    indexed.sort(key=lambda x: x[1][1])  # sort by priority value

    wt = [0]*len(burst_times)
    tat = [0]*len(burst_times)
```

```python
        tat = [0]*len(burst_times)

        current = 0
        for pid, (bt, pr) in indexed:
            wt[pid] = current
            tat[pid] = current + bt
            current += bt

        return wt, tat


    # ------------------- Round Robin -------------------
    def round_robin(burst_times, quantum):
        remaining = burst_times[:]
        n = len(burst_times)
        t = 0
        wt = [0]*n
        tat = [0]*n
        done = False

        while not done:
            done = True
            for i in range(n):
                if remaining[i] > 0:
                    done = False
                    if remaining[i] > quantum:
                        t += quantum
                        remaining[i] -= quantum
                    else:
                        t += remaining[i]
                        wt[i] = t - burst_times[i]
                        remaining[i] = 0

        for i in range(n):
            tat[i] = wt[i] + burst_times[i]

```

```
75
76
77  # ------------------- MAIN PROGRAM -------------------
78  if __name__ == "__main__":
79      print("----- CPU Scheduling Algorithms -----")
80      n = int(input("Enter number of processes: "))
81
82      burst = []
83      for i in range(n):
84          burst.append(int(input(f"Enter burst time for P{i+1}: ")))
85
86      print("\nFCFS Scheduling:")
87      wt, tat = fcfs(burst)
88      print("WT =", wt)
89      print("TAT =", tat)
90
91      print("\nSJF Scheduling:")
92      wt, tat = sjf(burst)
93      print("WT =", wt)
94      print("TAT =", tat)
95
96      print("\nPriority Scheduling:")
97      priority_list = []
98      for i in range(n):
99          priority_list.append(int(input(f"Enter priority for P{i+1}: ")))
100     wt, tat = priority_scheduling(burst, priority_list)
101     print("WT =", wt)
102     print("TAT =", tat)
103
104     print("\nRound Robin Scheduling:")
105     quantum = int(input("Enter time quantum: "))
106     wt, tat = round_robin(burst, quantum)
107     print("WT =", wt)
108     print("TAT =", tat)
```

Output :

```
----- CPU Scheduling Algorithms -----
Enter number of processes: 4
Enter burst time for P1: 5
Enter burst time for P2: 3
Enter burst time for P3: 8
Enter burst time for P4: 6

FCFS Scheduling:
WT = [0, 5, 8, 16]
TAT = [5, 8, 16, 22]

SJF Scheduling:
WT = [3, 0, 14, 8]
TAT = [8, 3, 22, 14]
```

```
TAT = [8, 3, 22, 14]

Priority Scheduling:
Enter priority for P1: 4
Enter priority for P2: 6
Enter priority for P3: 2
Enter priority for P4: 8
WT = [8, 13, 0, 16]
TAT = [13, 16, 8, 22]

Round Robin Scheduling:
Enter time quantum: 3
WT = [9, 3, 14, 14]
TAT = [14, 6, 22, 20]
harshkumarmishra16@Harshs-MacBook-Air lab sheet 4 %
```