

# **Lab Experiment Sheet - 04**



**Student Name: SURYANSH**

**Roll No. 2301730126**

**Course Code- ENCS351**

**Course Name – Operating system**

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

**GitHub Repository Link :**

**<https://github.com/Suryansh-Dhama/OS-Assignment.git>**

## **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
```

```
1 import subprocess
2
3 scripts = ['script1.py', 'script2.py', 'script3.py']
4
5 print("----- Batch Processing Started -----")
6
7 for script in scripts:
8     print(f"\nExecuting {script} ...")
9     subprocess.call(['python3', script])
10
11 print("\n----- Batch Processing Completed -----")
```

## Output :

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ⌂ ⌂ ⌂ ⌂ ⌂ X
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 or 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 or 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 or 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

```
1 import multiprocessing
2 import logging
3 import time
4
5 logging.basicConfig(
6     filename='system_log.txt',
7     level=logging.INFO,
8     format='%(asctime)s - %(processName)s - %(message)s'
9 )
10
11 def process_task(name):
12     logging.info(f"{name} started")
13     time.sleep(2)
14     logging.info(f"{name} terminated")
15
16 if __name__ == '__main__':
17     print("System Booting...\n")
18
19 p1 = multiprocessing.Process(target=process_task, args=("Process-1",))
20 p2 = multiprocessing.Process(target=process_task, args=("Process-2",))
21
22 p1.start()
23 p2.start()
24
25 p1.join()
26 p2.join()
27
28 print("\nSystem Shutdown.")
```

**Output :**

```

harshkumarmishra16@Harshs-MacBook-Air lab sheet 4 % /usr/bin/python3 "/Users/harshkumarmishra16/Documents/lab sheet 4/batch_processing.py"
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 or 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 or directory
----- Batch Processing Completed -----
● harshkumarmishra16@Harshs-MacBook-Air lab sheet 4 % /usr/bin/python3 "/Users/harshkumarmishra16/Documents/lab sheet 4/system_startup.py"
System Booting...

System Shutdown.
● harshkumarmishra16@Harshs-MacBook-Air lab sheet 4 %

```

Ln 1, Col 1 Spaces: 4 UTF-8 LF Plain Text Go Live

```

system_log.txt
1 2025-11-16 16:33:02,273 - Process-1 - Process-1 started
2 2025-11-16 16:33:02,273 - Process-2 - Process-2 started
3 2025-11-16 16:33:04,278 - Process-1 - Process-1 terminated
4 2025-11-16 16:33:04,279 - Process-2 - Process-2 terminated
5 2025-11-16 16:55:32,659 - Process-1 - Process-1 started
6 2025-11-16 16:55:32,659 - Process-2 - Process-2 started
7 2025-11-16 16:55:34,665 - Process-1 - Process-1 terminated
8 2025-11-16 16:55:34,665 - Process-2 - Process-2 terminated
9 2025-11-16 17:06:06,224 - Process-2 - Process-2 started
10 2025-11-16 17:06:06,224 - Process-1 - Process-1 started
11 2025-11-16 17:06:08,229 - Process-2 - Process-2 terminated
12 2025-11-16 17:06:08,229 - Process-1 - Process-1 terminated
13

```

### 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
<pre> 1     import os 2 3     r, w = os.pipe() 4     pid = os.fork() 5 6     if pid &gt; 0: 7         # Parent 8         os.close(r) 9         os.write(w, b"Hello from parent process via pipe!") 10        os.close(w) 11        os.wait() 12    else: 13        # Child 14        os.close(w) 15        data = os.read(r, 1024) 16        print("Child received:", data.decode()) 17        os.close(r) </pre>		

### Output :

```

$ python ipc_fork_exec.py
● 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

```
1 import os  
2 import platform  
3 import subprocess  
4  
5 ~ def detect_virtual_machine():  
6     print("----- VM Detection -----")  
7  
8     # Method 1: Check system manufacturer  
9     try:  
10         output = subprocess.check_output("dmidecode -s system-manufacturer", shell=True).decode().lower()  
11         if "vmware" in output or "virtualbox" in output or "qemu" in output:  
12             print("Running inside a Virtual Machine.")  
13             return  
14     except:  
15         pass  
16  
17     # Method 2: Check CPU virtualization flag  
18     cpufreq = ""  
19     try:  
20         cpufreq = open("/proc/cpuinfo").read().lower()  
21         if "hypervisor" in cpufreq:  
22             print("Hypervisor flag detected → VM environment.")  
23             return  
24     except:  
25         pass  
26  
27     # Method 3: Check platform metadata  
28     if "virtual" in platform.platform().lower():  
29         print("VM pattern found in platform data.")  
30         return  
31  
32     print("System appears to be running on a physical machine.")  
33  
34 detect_virtual_machine()
```

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

```
1  #!/bin/bash
2  echo "----- System Information -----"
3
4  echo "Kernel Version:"
5  uname -r
6
7  echo "User:"
8  whoami
9
10 echo "Hardware Info:"
11 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

```
1      # ----- FCFS -----
2  ↘  def fcfs(burst_times):
3      wt = [0] * len(burst_times)
4      tat = [0] * len(burst_times)
5
6      for i in range(1, len(burst_times)):
7          wt[i] = wt[i-1] + burst_times[i-1]
8
9      for i in range(len(burst_times)):
10         tat[i] = wt[i] + burst_times[i]
11
12     return wt, tat
13
14
15     # ----- SJF -----
16  ↘  def sjf(burst_times):
17      indexed = list(enumerate(burst_times))
18      indexed.sort(key=lambda x: x[1])
19
20      wt = [0]*len(burst_times)
21      tat = [0]*len(burst_times)
22
23      current = 0
24      for pid, bt in indexed:
25          wt[pid] = current
26          tat[pid] = current + bt
27          current += bt
28
29      return wt, tat
30
31
32     # ----- Priority -----
33  ↘  def priority_scheduling(burst_times, priority):
34      indexed = list(enumerate(zip(burst_times, priority)))
35      indexed.sort(key=lambda x: x[1][1]) # sort by priority value
36
37      wt = [0]*len(burst_times)
38      tat = [0]*len(burst_times)
```

```
38     tat = [0]*len(burst_times)
39
40     current = 0
41     for pid, (bt, pr) in indexed:
42         wt[pid] = current
43         tat[pid] = current + bt
44         current += bt
45
46     return wt, tat
47
48
49     # ----- Round Robin -----
50     def round_robin(burst_times, quantum):
51         remaining = burst_times[:]
52         n = len(burst_times)
53         t = 0
54         wt = [0]*n
55         tat = [0]*n
56         done = False
57
58         while not done:
59             done = True
60             for i in range(n):
61                 if remaining[i] > 0:
62                     done = False
63                     if remaining[i] > quantum:
64                         t += quantum
65                         remaining[i] -= quantum
66                     else:
67                         t += remaining[i]
68                         wt[i] = t - burst_times[i]
69                         remaining[i] = 0
70
71             for i in range(n):
72                 tat[i] = wt[i] + burst_times[i]
73
```

```
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 % █
```

Thank  
+ you +