

# KR MANGALAM UNIVERSITY

SOHNA ROAD, GURUGRAM



## OPERATING SYSTEM LAB FILE

COURSE CODE-ENCS351

**Submitted by**

Name: Aldrin Debbarma  
No.: 2401011482

**Submitted To**

Dr Satinder Pal Singh Roll

**Program:** BTech CSE  
**Section:** 6/F  
**:Session:** 2025/26  
**Date:** 02/12/2025

## 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])
```

The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER View:** Shows files in the current workspace, including `main.py`, `process_log.txt`, `README.md`, `service_dependency_log.txt`, `system_boot_log.txt`, and `system_shutdown_log.txt`.
- Code Editor:** Displays the `main.py` script with the provided Python code.
- TERMINAL View:** Shows the command `ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py` being run, followed by the output of the batch processing simulation. The output indicates that the script tried to execute three files sequentially but failed to find them, resulting in errors for each file.
- Bottom Status Bar:** Provides information about the current file (`main.py`), line (`Ln 14, Col 44`), and character count (`Spaces: 4`), along with other terminal-related settings like `BLACKBOX Autocomplete: ON` and `BLACKBOXAI: Open Chat`.

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

The screenshot shows the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left lists files: main.py, process\_log.txt, README.md, service\_dependency\_log.txt, system\_boot\_log.txt, system\_log.txt, and system\_shutdown\_log.txt. The main editor area displays the content of main.py:

```
6  logging.basicConfig(
7      filename='system_log.txt',
8      level=logging.INFO,
9      format='%(asctime)s - %(processName)s - %(message)s'
10 )
11
12 # Dummy task for child processes
13 def process_task(name):
14     logging.info(f'{name} started')
15     print(f'{name} started...')
16     time.sleep(2)           # simulate system work
17     logging.info(f'{name} terminated')
18     print(f'{name} terminated.')
19
20 if __name__ == "__main__":
21     print("System Booting...")
22     logging.info("System Boot Initiated")
23
24     # Create child processes
25     p1 = multiprocessing.Process(target=process_task, args=(("Process-1",))
26     p2 = multiprocessing.Process(target=process_task, args=(("Process-2",))
27
28     # Start processes
29     p1.start()
30     p2.start()
31
32     # Wait for processes to finish
33     p1.join()
34     p2.join()
35
36     logging.info("System Shutdown Completed")
37     print("\nSystem Shutdown.")
```

The terminal at the bottom shows the execution of the script:

```
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py
System Booting...

Process-2 started...
Process-1 started...
Process-1 terminated.
Process-2 terminated.

System Shutdown.
ashishyaduvanshi@Ashishs-MacBook-Air python %
```

At the bottom, the status bar shows: Ln 20, Col 24 Spaces: 4 UTF-8 LF Python % BLACKBOX Autocomplete: ON % BLACKBOX AI: Open Chat.

### **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)
```

The screenshot shows the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left lists files: main.py, process\_log.txt, README.md, service\_dependency\_log.txt, system\_boot\_log.txt, system\_log.txt, and system\_shutdown\_log.txt. The main editor window displays the content of main.py:

```
1 import multiprocessing
2 import os
3
4 def child_process(pipe_conn):
5     """Child reads data sent by Parent."""
6     message = pipe_conn.recv()      # Read message from parent
7     print(f"[Child] PID = {os.getpid()} received: {message}")
8     pipe_conn.close()
9
10
11 def parent_child_pipe():
12     parent_conn, child_conn = multiprocessing.Pipe() # Create a pipe
13
14     print("== PIPE IPC Simulation Started ==")
15
16     # Create child process
17     child = multiprocessing.Process(target=child_process, args=(child_conn,))
18
19     child.start()
20
21     # Parent sending message
22     msg = "Hello Child, this is Parent!"
23     print(f"[Parent] PID = {os.getpid()} sending: {msg}")
24     parent_conn.send(msg)
25
26     child.join()
27
28     print("\n== PIPE IPC Simulation Completed ==")
29
30
31 if __name__ == "__main__":
32     parent_child_pipe()
```

The terminal at the bottom shows the output of running the script:

```
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py
== PIPE IPC Simulation Started ==
[Parent] PID = 34538 sending: Hello Child, this is Parent!
[Child] PID = 34540 received: Hello Child, this is Parent!
== PIPE IPC Simulation Completed ==
ashishyaduvanshi@Ashishs-MacBook-Air python %
```

## 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'
```

The screenshot shows the Visual Studio Code interface. The left sidebar displays the file structure under 'EXPLORER' with files like 'main.py', 'process\_log.txt', 'README.md', 'service\_dependency\_log.txt', 'system\_boot\_log.txt', 'system\_log.txt', and 'system\_shutdown\_log.txt'. The main editor area shows a Python script named 'main.py' with the following code:

```
def detect_vm():
    if os.path.exists("/sys/class/dmi/id/product_name"):
        with open("/sys/class/dmi/id/product_name") as f:
            if any(x in name.lower() for x in ["virtual", "kvm", "vmware", "qemu", "hyper"]):
                return

    # 2. Check using systemd-detect-virt (Linux)
    try:
        result = subprocess.check_output(["systemd-detect-virt"], text=True).strip()
        print(f"systemd-detect-virt: {result}")
        if result != "none":
            print("> Running inside a Virtual Machine")
            return
    except Exception:
        pass

    # 3. Check virtualization flag (Intel/AMD VT)
    cpu_info = ""
    try:
        cpu_info = subprocess.check_output("lscpu", shell=True, text=True)
        if "Hypervisor vendor" in cpu_info:
            print("Hypervisor vendor detected - VM confirmed")
            return
    except:
        pass

    print("> No Virtual Machine detected")

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

Below the editor is a terminal window titled 'zsh' showing the command execution and output:

```
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py
zsh: command not found: python3
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py
== VM Detection Started ==

/bin/sh: lscpu: command not found
> No Virtual Machine detected
ashishyaduvanshi@Ashishs-MacBook-Air python %
```

The bottom status bar indicates the file is 'Ln 41, Col 24' with 'Spaces: 4' and 'UTF-8' encoding, and shows 'Python' and 'BLACKBOX Autocomplete: ON'.

## 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)

The screenshot shows a dark-themed code editor interface with the following details:

- EXPLORER:** Shows files in the 'PYTHON' folder: main.py, process\_log.txt, README.md, service\_dependency\_log.txt, system\_boot\_log.txt, system\_log.txt, and system\_shutdown\_log.txt.
- MAIN PANE:** Displays the content of 'main.py'. The code defines four functions: fcfs, sjf, priority, and round\_robin. The fcfs function calculates waiting time (wt) and turn-around time (tat) for processes based on burst times (bt). The sjf function does the same for SJF scheduling. The priority function prints a header and then processes data from a file named 'process\_log.txt'. The round\_robin function performs Round Robin scheduling with a quantum of 3 units.
- TERMINAL:** Shows the command 'ashishyaduvanshi@ashishs-MacBook-Air python % python3 main.py' being run.
- OUTPUT:** Displays the results of the execution:
  - Priority Scheduling:** PID 2, BT 8, PR 1, WT 0, TAT 8; PID 1, BT 6, PR 2, WT 8, TAT 14; PID 4, BT 3, PR 2, WT 14, TAT 17; PID 3, BT 7, PR 3, WT 17, TAT 24.
    - Average WT = 9.75
    - Average TAT = 15.75
  - Round Robin Scheduling:** PID 1, BT 6, MT 11, TAT 17; PID 2, BT 8, MT 15, TAT 23; PID 3, BT 7, MT 17, TAT 24; PID 4, BT 3, MT 12, TAT 15.
    - Average WT = 13.75
    - Average TAT = 19.75