

Nidhi Kumari

2401011484

LAB ASSIGNMENT-04

Experiment Title: System Calls, VM Detection, and File System Operations using Python

Task 1: Batch Processing Simulation (Python)

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

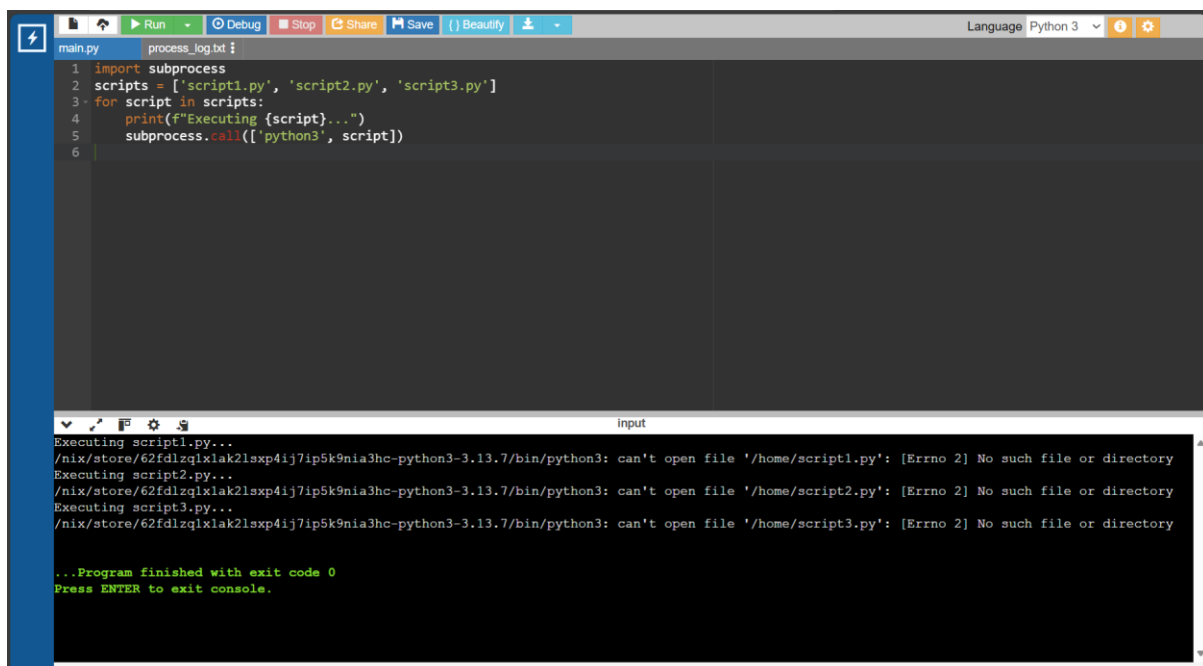
Implementation:

```
import subprocess

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

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

Output:



```
main.py process_log.txt
1 import subprocess
2 scripts = ['script1.py', 'script2.py', 'script3.py']
3 for script in scripts:
4     print(f"Executing {script}...")
5     subprocess.call(['python3', script])
6

input
Executing script1.py...
/nix/store/62fdlzlxlak2lasp4ij7ip5k9nia3hc-python3-3.13.7/bin/python3: can't open file '/home/script1.py': [Errno 2] No such file or directory
Executing script2.py...
/nix/store/62fdlzlxlak2lasp4ij7ip5k9nia3hc-python3-3.13.7/bin/python3: can't open file '/home/script2.py': [Errno 2] No such file or directory
Executing script3.py...
/nix/store/62fdlzlxlak2lasp4ij7ip5k9nia3hc-python3-3.13.7/bin/python3: can't open file '/home/script3.py': [Errno 2] No such file or directory

...Program finished with exit code 0
Press ENTER to exit console.
```

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.

Implementation:

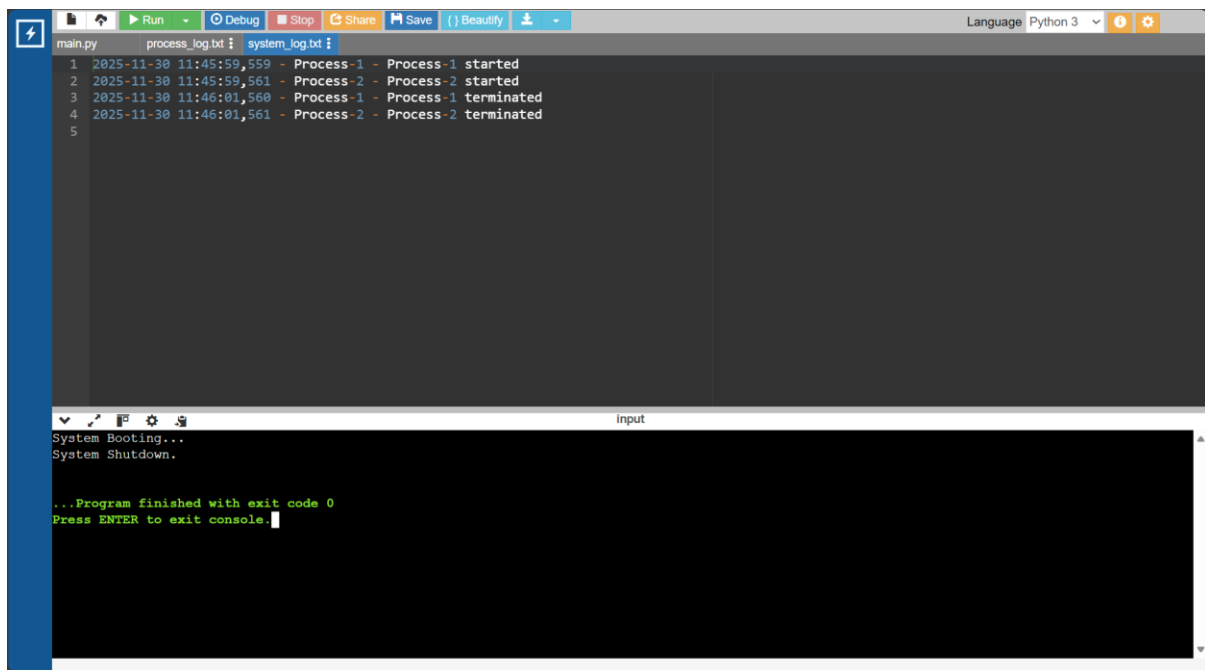
```
import multiprocessing
import logging
import time

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.")
```

Output:



The screenshot shows a Python IDE interface. The top toolbar includes icons for Run, Debug, Stop, Share, Save, and Beautify. The language is set to Python 3. The editor displays a file named `main.py` with the following content:

```
1 2025-11-30 11:45:59,559 - Process-1 - Process-1 started
2 2025-11-30 11:45:59,561 - Process-2 - Process-2 started
3 2025-11-30 11:46:01,560 - Process-1 - Process-1 terminated
4 2025-11-30 11:46:01,561 - Process-2 - Process-2 terminated
5
```

Below the editor, there is a console window titled "input" showing the following output:

```
System Booting...
System Shutdown.

...Program finished with exit code 0
Press ENTER to exit console.
```

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.

Implementation:

```
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)
```

Output:



The screenshot shows a code editor with a Python script and its output. The script is a parent-child process communication example using pipes. The parent process forks a child process, writes "Hello from parent" to the pipe, and then waits for the child to finish. The child process reads the message from the pipe and prints it. The output shows the child received the message and the program finished with exit code 0.

```
1 import os
2 r, w = os.pipe()
3 pid = os.fork()
4 if pid > 0:
5     os.close(r)
6     os.write(w, b"Hello from parent")
7     os.close(w)
8     os.wait()
9 else:
10    os.close(w)
11    message = os.read(r, 1024)
12    print("Child received:", message.decode())
13    os.close(r)
14
```

Child received: Hello from parent

...Program finished with exit code 0
Press ENTER to exit console.

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.

Implementation:

```
#!/bin/bash

echo "Kernel Version:"

uname -r

echo "User:"

whoami

echo "Hardware Info:"

lscpu | grep 'Virtualization'
```

Python Script:

```
import os
```

```
import subprocess
```

```
def check_dmi():
```

```
    """Check system DMI data for known VM identifiers."""
```

```
    vm_signatures = ["virtual", "vmware", "kvm", "qemu", "hyper-v", "xen"]
```

```
    try:
```

```
        output = subprocess.check_output(["sudo", "dmidecode"],
```

```
            stderr=subprocess.DEVNULL).decode().lower()
```

```
        return any(sig in output for sig in vm_signatures)
```

```
    except:
```

```
        return False
```

```
def check_cpu_flags():
```

```
    """Check CPU flags for hypervisor bit."""
```

```
    try:
```

```
        with open("/proc/cpuinfo") as f:
```

```
            data = f.read().lower()
```

```
            return "hypervisor" in data
```

```
    except:
```

```
        return False
```

```
def check_mac_address():
```

```
    """Check if the MAC address belongs to a VM vendor."""
```

```
    vm_mac_prefixes = [
```

```
        "00:05:69", "00:0C:29", "00:1C:14", # VMware
```

```
        "08:00:27", # VirtualBox
```

```
        "52:54:00", # QEMU / KVM
```

```
        "00:15:5D", # Hyper-V
```

```
    ]
```

```
    try:
```

```
        output = subprocess.check_output(["ip", "link"]).decode().lower()
```

```

        for prefix in vm_mac_prefixes:
            if prefix.lower() in output:
                return True
    except:
        pass
    return False

def detect_vm():
    print("\n--- Virtual Machine Detection ---")
    dmi = check_dmi()
    hypervisor_flag = check_cpu_flags()
    mac_vm = check_mac_address()
    if dmi or hypervisor_flag or mac_vm:
        print("This system appears to be running inside a VIRTUAL MACHINE.")
    else:
        print("This system appears to be running on BARE METAL hardware.")
    print("\nDetails:")
    print(f"DMI-based detection: {dmi}")
    print(f"CPU hypervisor flag: {hypervisor_flag}")
    print(f"MAC address virtual: {mac_vm}")

if __name__ == "__main__":
    detect_vm()

```

Output:

Task 5: CPU Scheduling Algorithms

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

Implementation:

```
"""FCFS Scheduling"""
```

```
def fcfs(processes):  
    processes.sort(key=lambda x: x['arrival'])  
    time = 0  
    for p in processes:  
        if time < p['arrival']:  
            time = p['arrival']  
        p['wt'] = time - p['arrival']  
        time += p['burst']  
        p['tat'] = p['wt'] + p['burst']  
    return processes
```

```
"""SJF Scheduling"""
```

```
def sjf(processes):  
    processes = sorted(processes, key=lambda x: x['arrival'])  
    completed, time = 0, 0  
    n = len(processes)  
    while completed < n:  
        available = [p for p in processes if p['arrival'] <= time and 'done' not in p]  
        if not available:  
            time += 1  
            continue  
        p = min(available, key=lambda x: x['burst'])  
        p['wt'] = time - p['arrival']  
        time += p['burst']  
        p['tat'] = p['wt'] + p['burst']
```

```
p['done'] = True
completed += 1
return processes
```

```
"""Round Robin"""
```

```
def round_robin(processes, quantum):
    from collections import deque
    q = deque()
    time = 0
    remaining = {p['pid']: p['burst'] for p in processes}
    processes.sort(key=lambda x: x['arrival'])
    i = 0
    completed = 0
    n = len(processes)
    while completed < n:
        while i < n and processes[i]['arrival'] <= time:
            q.append(processes[i])
            i += 1
        if not q:
            time = processes[i]['arrival']
            continue
        p = q.popleft()
        exec_time = min(quantum, remaining[p['pid']])
        remaining[p['pid']] -= exec_time
        time += exec_time
        while i < n and processes[i]['arrival'] <= time:
            q.append(processes[i])
            i += 1
        if remaining[p['pid']] == 0:
            p['tat'] = time - p['arrival']
            p['wt'] = p['tat'] - p['burst']
```



```

        completed += 1
    else:
        q.append(p)
    return processes

```

"""Priority Scheduling"""

```

def priority_scheduling(processes):
    time = 0
    completed = 0
    n = len(processes)
    processes.sort(key=lambda x: x['arrival'])
    while completed < n:
        available = [p for p in processes if p['arrival'] <= time and 'done' not in p]
        if not available:
            time += 1
            continue
        p = min(available, key=lambda x: x['priority'])
        p['wt'] = time - p['arrival']
        time += p['burst']
        p['tat'] = p['wt'] + p['burst']
        p['done'] = True
        completed += 1
    return processes

```

```

processes = [
    {'pid': 1, 'arrival': 0, 'burst': 5, 'priority': 2},
    {'pid': 2, 'arrival': 1, 'burst': 3, 'priority': 1},
    {'pid': 3, 'arrival': 2, 'burst': 8, 'priority': 4},
    {'pid': 4, 'arrival': 3, 'burst': 6, 'priority': 3},
]

```

```

import copy

print("\n--- FCFS ---")

for p in fcfs(copy.deepcopy(processes)):
    print(p)

print("\n--- SJF ---")

for p in sjf(copy.deepcopy(processes)):
    print(p)

print("\n--- Round Robin (Q=2) ---")

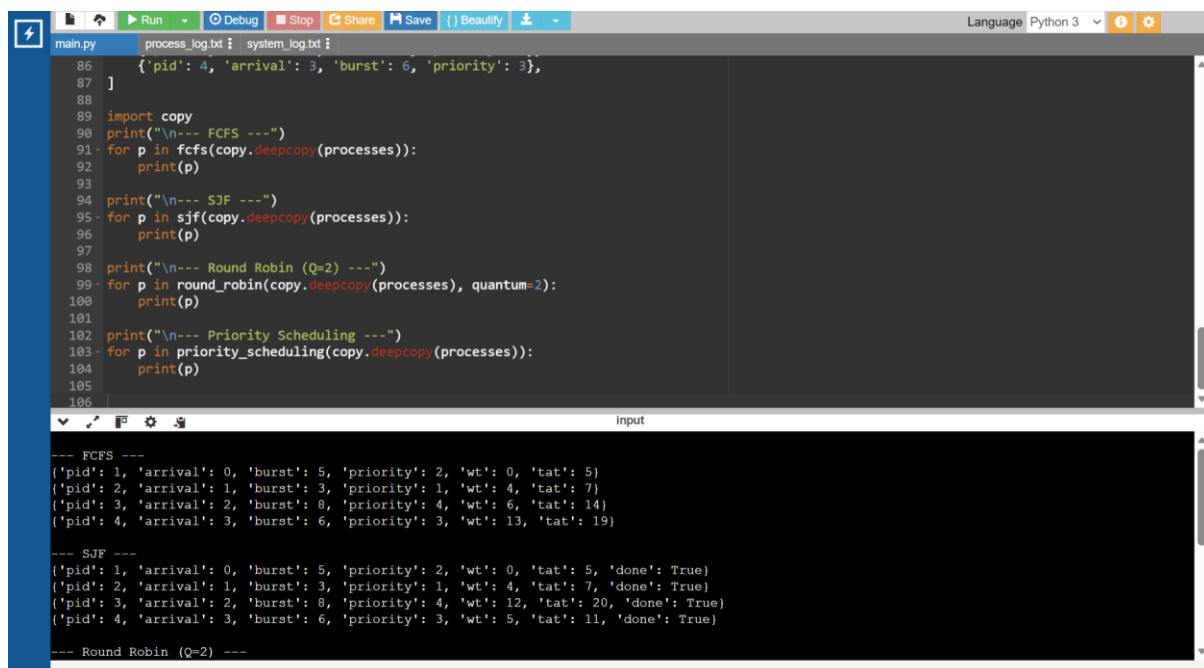
for p in round_robin(copy.deepcopy(processes), quantum=2):
    print(p)

print("\n--- Priority Scheduling ---")

for p in priority_scheduling(copy.deepcopy(processes)):
    print(p)

```

Output:



```

main.py process_log.txt system_log.txt
86     {'pid': 4, 'arrival': 3, 'burst': 6, 'priority': 3},
87 ]
88
89 import copy
90 print("\n--- FCFS ---")
91 for p in fcfs(copy.deepcopy(processes)):
92     print(p)
93
94 print("\n--- SJF ---")
95 for p in sjf(copy.deepcopy(processes)):
96     print(p)
97
98 print("\n--- Round Robin (Q=2) ---")
99 for p in round_robin(copy.deepcopy(processes), quantum=2):
100     print(p)
101
102 print("\n--- Priority Scheduling ---")
103 for p in priority_scheduling(copy.deepcopy(processes)):
104     print(p)
105
106
input
--- FCFS ---
({'pid': 1, 'arrival': 0, 'burst': 5, 'priority': 2, 'wt': 0, 'tat': 5})
({'pid': 2, 'arrival': 1, 'burst': 3, 'priority': 1, 'wt': 4, 'tat': 7})
({'pid': 3, 'arrival': 2, 'burst': 8, 'priority': 4, 'wt': 6, 'tat': 14})
({'pid': 4, 'arrival': 3, 'burst': 6, 'priority': 3, 'wt': 13, 'tat': 19})
--- SJF ---
({'pid': 1, 'arrival': 0, 'burst': 5, 'priority': 2, 'wt': 0, 'tat': 5, 'done': True})
({'pid': 2, 'arrival': 1, 'burst': 3, 'priority': 1, 'wt': 4, 'tat': 7, 'done': True})
({'pid': 3, 'arrival': 2, 'burst': 8, 'priority': 4, 'wt': 12, 'tat': 20, 'done': True})
({'pid': 4, 'arrival': 3, 'burst': 6, 'priority': 3, 'wt': 5, 'tat': 11, 'done': True})
--- Round Robin (Q=2) ---

```