K. R. Mangalam University

School of Engineering and Technology

Operating System Lab File

Course Code: ENCS351

Submitted by

Name: Jatin Bisht

Roll No.: 2301010360

Program:

BTech CSE

Section: F

Submitted To

Dr. Satinder

ASSIGNMENT – 1

Process Creation and Management Using Python OS Module

TASK 1 — Process Creation Utility

Aim:

Create N child processes using os.fork().
Each child prints:

- PID
- Parent PID
- Custom Message

Parent waits for all children.

CODE :-

```python
import os
import time

def create_children(n):
    for i in range(n):
        pid = os.fork()

        if pid == 0:  # Child
            print(f"[Child] PID: {os.getpid()}, PPID: {os.getppid()}, Message: Child {i} running")
            os._exit(0)

    # Parent waits for all children
    for i in range(n):
        os.wait()
    print("[Parent] All children have finished.")

if __name__ == "__main__":
    create_children(3)
```

Output :-

```
[Child] PID: 6807, PPID: 6803, Message: Child 0 running
[Child] PID: 6809, PPID: 6803, Message: Child 2 running
[Child] PID: 6808, PPID: 6803, Message: Child 1 running
[Parent] All children have finished.


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

TASK 2 — Command Execution Using exec()

Aim:

Modify Task 1 so each child executes *Linux commands* like ls, date, ps.

Code :-

```python
import os

commands = [
    ["ls", "-l"],
    ["date"],
    ["ps", "-e"]
]

for i, cmd in enumerate(commands):
    pid = os.fork()

    if pid == 0:  # Child
        print(f"\n[Child {i}] Executing command: {' '.join(cmd)}")
        os.execvp(cmd[0], cmd)

# Parent waits
for _ in commands:
    os.wait()

print("\n[Parent] All commands executed.")
```

Output :-

```
[Child 0] Executing command: ls -l

[Child 1] Executing command: date

[Child 2] Executing command: ps -e
Sun Nov 23 03:56:12 PM UTC 2025
total 4
-rw-r--r-- 1 runner43 runner43 371 Nov 23 15:56 main.py
    PID TTY          TIME CMD
      1 pts/0    00:00:00 cinit
     37 pts/2    00:00:00 bash
   1487 ?        00:00:15 a.out
   2286 ?        00:00:00 a.out
   2804 ?        00:00:55 a.out
   4325 ?        00:00:00 python3
   5191 ?        00:00:00 a.out
   5317 ?        00:00:00 tinit
   5318 ?        00:00:00 a.out
   5471 pts/1    00:00:00 bash
   5518 pts/1    00:00:00 sh
   5519 pts/1    00:00:00 tinit
   5520 pts/3    00:00:00 python3
   5526 pts/3    00:00:00 ps

[Parent] All commands executed.
```

TASK 3 — Zombie & Orphan Processes

Zombie Process

A zombie occurs when:

- Child exits
- Parent does not call wait()

Code :-

```python
import os
import time
pid = os.fork()
if pid == 0:
    print(f"[Child] PID: {os.getpid()} exiting…")
    os._exit(0)
else:
    print("[Parent] Not calling wait(), child becomes zombie.")
    time.sleep(20)
```

Output :-

```
[Parent] Not calling wait(), child becomes zombie.
[Child] PID: 861 exiting…
h


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

TASK 5 — Process Prioritization (nice values)

Aim:

Create CPU-intensive processes with different nice() values. Observe scheduling difference.

Code :-

```python
import os
import time
def cpu_task():
    x = 0
    for i in range(50_000_00):
        x += 1
nice_values = [0, 5, 10]
for n in nice_values:
    pid = os.fork()
    if pid == 0:
        os.nice(n)
        print(f"[Child] PID: {os.getpid()} Nice: {n}")
        cpu_task()
        os._exit(0)

# Parent waits
for _ in nice_values:
    os.wait()
print("All prioritized processes completed.")


```

Output :-

```
[Child] PID: 3543 Nice: 5
[Child] PID: 3542 Nice: 0
[Child] PID: 3544 Nice: 10
All prioritized processes completed.


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

ASSIGNMENT-2

System Startup, Process Creation, and Termination Simulation in Python

Sub-Tasks:

1. **Sub-Task 1:** Initialize the logging configuration to capture timestamped messages.
2. **Sub-Task 2:** Define a function that simulates a process task (e.g., sleep for 2 seconds).
3. **Sub-Task 3:** Create at least two processes and start them concurrently.
4. **Sub-Task 4:** Ensure proper termination and joining of processes, and verify the output in the log file.

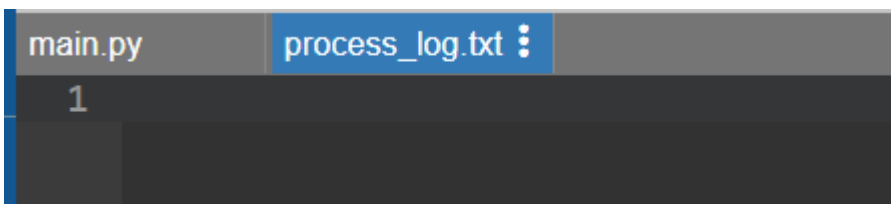
Sub-Task 1: Initialize the logging configuration

Objective: Set up the logging system to log messages with timestamps and process names.

Code :-

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

Output :-



Sub-Task 2: Define a function that simulates a process task

Objective: Write a function that mimics the work of a system process.

Code: -

```python
import time
# Dummy function to simulate a task
def system_process(task_name):
    logging.info(f"{task_name} started")
    time.sleep(2)  # Simulate task delay
    logging.info(f"{task_name} ended")

```

Sub-Task 3: Create at least two processes and start them concurrently

Objective: Use the multiprocessing module to initiate parallel tasks.

Code: -

```python
import multiprocessing
if __name__ == '__main__':
    print("System Starting...")
    # Create processes
    p1 = multiprocessing.Process(target=system_process, args=('Process-1',))
    p2 = multiprocessing.Process(target=system_process, args=('Process-2',))
    # Start processes
    p1.start()
    p2.start()

```

Sub-Task 4: Ensure proper termination and verify logs

Objective: Wait for processes to complete and confirm the shutdown.

```
1       p1.join()
2       p2.join()
3       print("System Shutdown.")
4
5
6
```

Output: -

```
System Starting...
System Shutdown.


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

Assignment -3

Simulation of File Allocation, Memory Management, and Scheduling in Python

Assignment Tasks:

Task 1: CPU Scheduling with Gantt Chart

Code: -

```python
# Priority Scheduling Simulation
processes = []
n = int(input("Enter number of processes: "))

for i in range(n):
    bt = int(input(f"Enter Burst Time for P{i+1}: "))
    pr = int(input(f"Enter Priority (lower number = higher priority) for P{i+1}: "))
    processes.append((i+1, bt, pr))
processes.sort(key=lambda x: x[2])
wt = 0
total_wt = 0
total_tt = 0
print("\nPriority Scheduling:")
print("PID\tBT\tPriority\tWT\tTAT")
for pid, bt, pr in processes:
    tat = wt + bt
    print(f"{pid}\t{bt}\t{pr}\t\t{wt}\t{tat}")
    total_wt += wt
    total_tt += tat
    wt += bt
print(f"Average Waiting Time: {total_wt / n}")
print(f"Average Turnaround Time: {total_tt / n}")

```

Output: -

```
Enter number of processes: 3
Enter Burst Time for P1: 5
Enter Priority (lower number = higher priority) for P1: 1
Enter Burst Time for P2: 3
Enter Priority (lower number = higher priority) for P2: 3
Enter Burst Time for P3: 1
Enter Priority (lower number = higher priority) for P3: 2

Priority Scheduling:
PID     BT      Priority        WT      TAT
1       5       1               0       5
3       1       2               5       6
2       3       3               6       9
Average Waiting Time: 3.6666666666666665
Average Turnaround Time: 6.666666666666667
```

Task 2: Sequential File Allocation

Code: -

```python
total_blocks = int(input("Enter total number of blocks: "))
block_status = [0] * total_blocks

n = int(input("Enter number of files: "))
for i in range(n):
    start = int(input(f"Enter starting block for file {i+1}: "))
    length = int(input(f"Enter length of file {i+1}: "))
    allocated = True
    for j in range(start, start + length):
        if j >= total_blocks or block_status[j] == 1:
            allocated = False
            break
    if allocated:
        for j in range(start, start + length):
            block_status[j] = 1
        print(f"File {i+1} allocated from block {start} to {start + length - 1}")
    else:
        print(f"File {i+1} cannot be allocated.")
```

Output: -

```
Enter total number of blocks: 10
Enter number of files: 2
Enter starting block for file 1: 2
Enter length of file 1: 4
File 1 allocated from block 2 to 5
Enter starting block for file 2: 4
Enter length of file 2: 5
File 2 cannot be allocated.
```

Task 3: Indexed File Allocation

Code: -

```python
total_blocks = int(input("Enter total number of blocks: "))
block_status = [0] * total_blocks
n = int(input("Enter number of files: "))
for i in range(n):
    index = int(input(f"Enter index block for file {i+1}: "))
    if block_status[index] == 1:
        print("Index block already allocated.")
        continue
    count = int(input("Enter number of data blocks: "))
    data_blocks = list(map(int, input("Enter block numbers: ").split()))
    if any(block_status[blk] == 1 for blk in data_blocks) or len(data_blocks) != count:
        print("Block(s) already allocated or invalid input.")
        continue
    block_status[index] = 1
    for blk in data_blocks:
        block_status[blk] = 1
    print(f"File {i+1} allocated with index block {index} -> {data_blocks}")

```

Output: -

```
Enter total number of blocks: 10
Enter number of files: 2
Enter index block for file 1: 3
Enter number of data blocks: 2
Enter block numbers: 4 5 6
Block(s) already allocated or invalid input.
Enter index block for file 2: 4
Enter number of data blocks: 3
Enter block numbers: 2
Block(s) already allocated or invalid input.
```

Task 4: Contiguous Memory Allocation

Code: -

```python
def allocate_memory(strategy):
    partitions = list(map(int, input("Enter partition sizes: ").split()))
    processes = list(map(int, input("Enter process sizes: ").split()))
    allocation = [-1] * len(processes)
    for i, psize in enumerate(processes):
        idx = -1
        if strategy == "first":
            for j, part in enumerate(partitions):
                if part >= psize:
                    idx = j
                    break
        elif strategy == "best":
            best_fit = float("inf")
            for j, part in enumerate(partitions):
                if part >= psize and part < best_fit:
                    best_fit = part
                    idx = j
        elif strategy == "worst":
            worst_fit = -1
            for j, part in enumerate(partitions):
                if part >= psize and part > worst_fit:
                    worst_fit = part
                    idx = j
        if idx != -1:
            allocation[i] = idx
            partitions[idx] -= psize
    print(f"\n{strategy.upper()} FIT ALLOCATION:")
```

Output: -

Firstfit: -

```
Enter partition sizes: 100 300 400 500 600
Enter process sizes: 212 112 417 235 231

FIRST FIT ALLOCATION:
Process 1 allocated in Partition 2
Process 2 allocated in Partition 3
Process 3 allocated in Partition 4
Process 4 allocated in Partition 3
Process 5 allocated in Partition 5
```

Best fit: -

```
Enter partition sizes: 200 100 300 500
Enter process sizes: 150 213 399 200

BEST FIT ALLOCATION:
Process 1 allocated in Partition 1
Process 2 allocated in Partition 3
Process 3 allocated in Partition 4
Process 4 cannot be allocated
```

Worst fit: -

```
Enter partition sizes: 200 300 100 500
Enter process sizes: 212 112 345 231

WORST FIT ALLOCATION:
Process 1 allocated in Partition 4
Process 2 allocated in Partition 2
Process 3 cannot be allocated
Process 4 allocated in Partition 4
```

Task 5: MFT & MVT Memory Management

Code: -

```python
def MFT():
    mem_size = int(input("Enter total memory size: "))
    part_size = int(input("Enter partition size: "))
    n = int(input("Enter number of processes: "))
    partitions = mem_size // part_size
    print(f"Memory divided into {partitions} partitions")
    for i in range(n):
        psize = int(input(f"Enter size of Process {i+1}: "))
        if psize <= part_size:
            print(f"Process {i+1} allocated.")
        else:
            print(f"Process {i+1} too large for fixed partition.")

def MVT():
    mem_size = int(input("Enter total memory size: "))
    n = int(input("Enter number of processes: "))

    for i in range(n):
        psize = int(input(f"Enter size of Process {i+1}: "))
        if psize <= mem_size:
            print(f"Process {i+1} allocated.")
            mem_size -= psize
        else:
            print(f"Process {i+1} cannot be allocated. Not enough memory.")
print("MFT Simulation:")
MFT()
print("\nMVT Simulation:")
```

Output: -

```
MFT Simulation:
Enter total memory size: 1000
Enter partition size: 200
Enter number of processes: 3
Memory divided into 5 partitions
Enter size of Process 1: 250
Process 1 too large for fixed partition.
Enter size of Process 2: 150
Process 2 allocated.
Enter size of Process 3: 200
Process 3 allocated.
```

ASSIGNMENT – 4

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

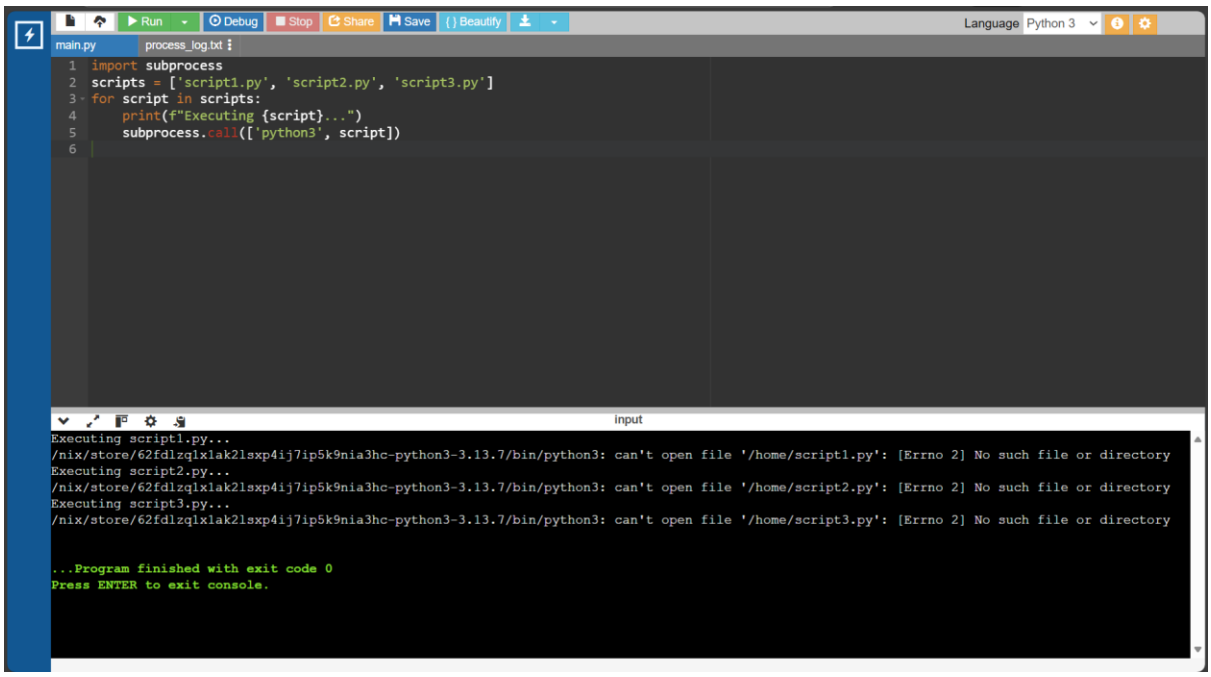
Sub Tasks

Task 1: Batch Processing Simulation (Python)

Implementation:

```
import subprocess
scripts = ['script1.py', 'script2.py', 'script3.py']
for script in scripts:
    print(f"Executing {script}...")
    subprocess.call(['python3', script])
```

Output: -



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:

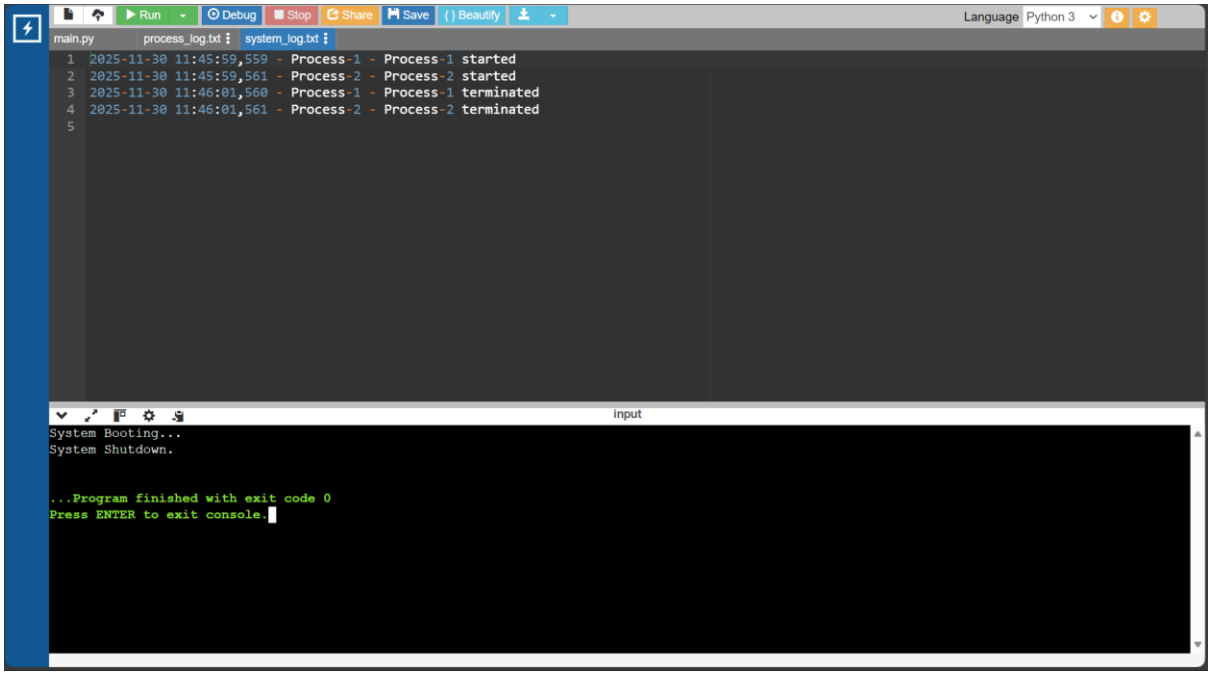
```
import multiprocessing
import logging
import time

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: -



Task 3: System Calls and IPC (Python - fork, exec, pipe)

Use system calls (fork(), exec(), wait()) and implement basic Inter-Process

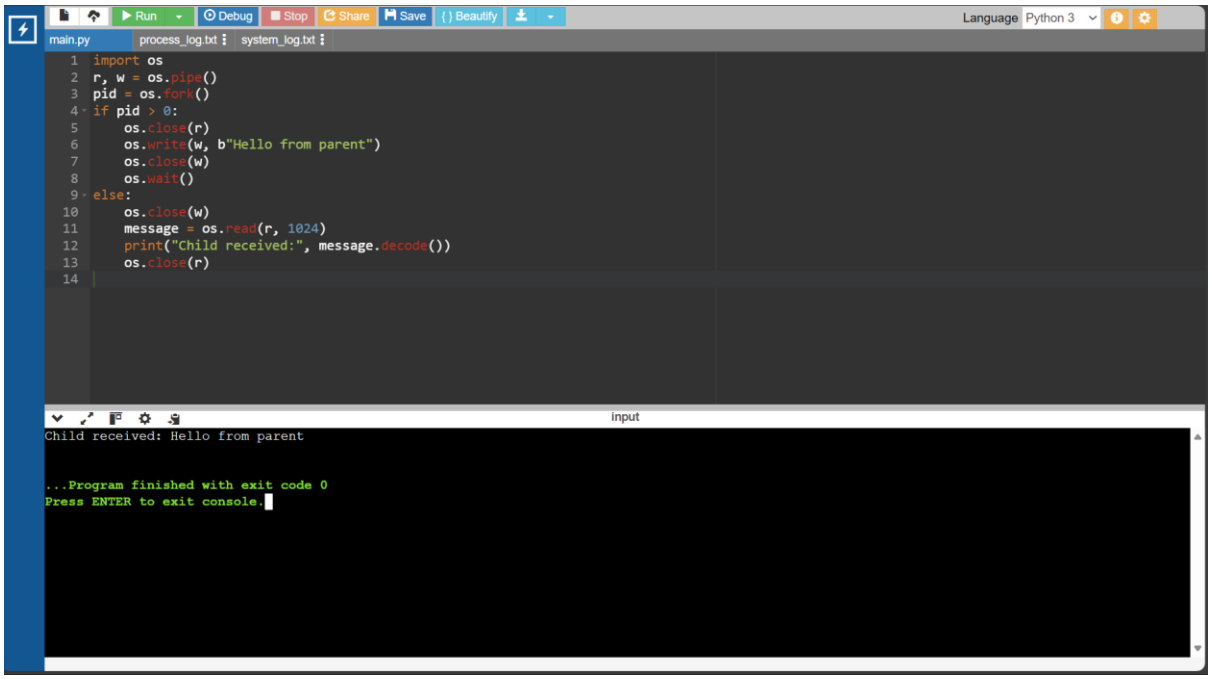
Communication using pipes in C or Python.

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: -



Task 5: CPU Scheduling Algorithms

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

Implementation:

```
import pandas as pd
def compute_wt_tat(df):
    df['TAT'] = df['CT'] - df['AT']   # Turnaround Time
    df['WT'] = df['TAT'] - df['BT']   # Waiting Time
    return df
def fcfs(processes):
    df = pd.DataFrame(processes)
    df = df.sort_values(by=['AT']).reset_index(drop=True)
```

```
    CT = []
    t = 0
    for i in range(len(df)):
        if t < df.loc[i, 'AT']:
            t = df.loc[i, 'AT']
        t += df.loc[i, 'BT']
        CT.append(t)

    df['CT'] = CT
    df = compute_wt_tat(df)
    return df
def sjf(processes):
    df = pd.DataFrame(processes)
    df = df.sort_values(by=['AT']).reset_index(drop=True)

    completed = []
    t = 0
    CT = [0] * len(df)
    done = 0

    while done < len(df):
        ready = df[(df['AT'] <= t) & (~df.index.isin(completed))]
        if len(ready) == 0:
            t += 1
            continue

        idx = ready['BT'].idxmin()
        t += df.loc[idx, 'BT']
        CT[idx] = t
```

```
        completed.append(idx)
        done += 1

    df['CT'] = CT
    df = compute_wt_tat(df)
    return df
def priority_scheduling(processes):
    df = pd.DataFrame(processes)
    df = df.sort_values(by=['AT']).reset_index(drop=True)

    completed = []
    t = 0
    CT = [0] * len(df)
    done = 0

    while done < len(df):
        ready = df[(df['AT'] <= t) & (~df.index.isin(completed))]
        if len(ready) == 0:
            t += 1
            continue

        idx = ready['Priority'].idxmin()  # lower value = higher priority
        t += df.loc[idx, 'BT']
        CT[idx] = t
        completed.append(idx)
        done += 1

    df['CT'] = CT
    df = compute_wt_tat(df)
```

```
    return df
def round_robin(processes, quantum):
    df = pd.DataFrame(processes)
    df = df.sort_values(by=['AT']).reset_index(drop=True)

    rt = df['BT'].tolist()
    CT = [0] * len(df)
    t = 0
    queue = []
    visited = [False] * len(df)

    while True:
        for i in range(len(df)):
            if df.loc[i, 'AT'] <= t and not visited[i]:
                queue.append(i)
                visited[i] = True
        if not queue:
            t += 1
            continue
        idx = queue.pop(0)
        if rt[idx] > quantum:
            t += quantum
            rt[idx] -= quantum
        else:
            t += rt[idx]
            CT[idx] = t
            rt[idx] = 0
        for i in range(len(df)):
            if df.loc[i, 'AT'] <= t and not visited[i] and rt[i] > 0:
```

```
            queue.append(i)
            visited[i] = True
        if rt[idx] > 0:
            queue.append(idx)
        if all(r == 0 for r in rt):
            break
    df['CT'] = CT
    df = compute_wt_tat(df)
    return df
def round_robin(processes, quantum):
    df = pd.DataFrame(processes)
    df = df.sort_values(by=['AT']).reset_index(drop=True)
    rt = df['BT'].tolist()
    CT = [0] * len(df)
    t = 0
    queue = []
    visited = [False] * len(df)
    while True:
        for i in range(len(df)):
            if df.loc[i, 'AT'] <= t and not visited[i]:
                queue.append(i)
                visited[i] = True
        if not queue:
            t += 1
            continue
        idx = queue.pop(0)
        if rt[idx] > quantum:
            t += quantum
            rt[idx] -= quantum
```

```
        else:
            t += rt[idx]
            CT[idx] = t
            rt[idx] = 0
        for i in range(len(df)):
            if df.loc[i, 'AT'] <= t and not visited[i] and rt[i] > 0:
                queue.append(i)
                visited[i] = True
        if rt[idx] > 0:
            queue.append(idx)

        if all(r == 0 for r in rt):
            break
    df['CT'] = CT
    df = compute_wt_tat(df)
    return df
process_list = [
    {'PID': 'P1', 'AT': 0, 'BT': 6, 'Priority': 2},
    {'PID': 'P2', 'AT': 1, 'BT': 8, 'Priority': 1},
    {'PID': 'P3', 'AT': 2, 'BT': 7, 'Priority': 3},
    {'PID': 'P4', 'AT': 3, 'BT': 3, 'Priority': 2}
]
print("🔹 FCFS Scheduling")
display(fcfs(process_list))

print("🔹 SJF Scheduling")
display(sjf(process_list))
print("🔹 Priority Scheduling")
display(priority_scheduling(process_list))
```

```
print(" 🔹 Round Robin (Q=3)")
display(round_robin(process_list, quantum=3))
```

Output: -

🔹 FCFS Scheduling

	PID	AT	BT	Priority	CT	TAT	WT
0	P1	0	6	2	6	6	0
1	P2	1	8	1	14	13	5
2	P3	2	7	3	21	19	12
3	P4	3	3	2	24	21	18

🔹 SJF Scheduling

	PID	AT	BT	Priority	CT	TAT	WT
0	P1	0	6	2	6	6	0
1	P2	1	8	1	24	23	15
2	P3	2	7	3	16	14	7
3	P4	3	3	2	9	6	3

🔹 Priority Scheduling

	PID	AT	BT	Priority	CT	TAT	WT
0	P1	0	6	2	6	6	0
1	P2	1	8	1	14	13	5
2	P3	2	7	3	24	22	15
3	P4	3	3	2	17	14	11

🔹 Round Robin (Q=3)

	PID	AT	BT	Priority	CT	TAT	WT
0	P1	0	6	2	15	15	9
1	P2	1	8	1	23	22	14
2	P3	2	7	3	24	22	15
3	P4	3	3	2	12	9	6