**Lab Assignment 3**

*Name: Vivek Sapkal*

*Roll No.: B22AI066*

## **Concept Simulated:**

* + The code simulates the scheduling of processes in an operating system. It involves three main components: the long-term scheduler, the short-term scheduler, and the waiting process handler.

## **How the Code Simulates it:**

* + Multiprocessing: It employs multiprocessing to simulate concurrent execution of processes.
  + Process Representation: Processes are represented using Process Control Blocks (PCBs) with attributes such as ID and state.
  + Randomization: Random probabilities are used to mimic process completion, interruption, and waiting for I/O.
  + Long-term Scheduler: Sorts processes based on priority and places them in the ready queue.
  + Short-term Scheduler: Executes processes from the ready queue, simulating their execution as separate processes.
  + Waiting Process Handler: Manages processes waiting for I/O, moving them between the I/O waiting queue and the ready queue.
  + When the code is run it starts simulation showing on the terminal which process is running, interrupted, waiting for I/O or completed.

## **Code and Explanation:**

### **PCB Class and Imports:**

import multiprocessing

import time

import random

class PCB:

def \_\_init\_\_(self, process\_id, program\_counter, memory\_limit):

self.process\_id = process\_id

self.program\_counter = program\_counter

self.memory\_limit = memory\_limit

self.state = "NEW" # Initial state

self.priority = random.randint(1, 10) # Assign a random priority

def set\_state(self, new\_state):

self.state = new\_state

1. Import lines import necessary modules for multiprocessing (multiprocessing), time-related functions (time), and random number generation (random).
2. This defines a class PCB (Process Control Block) representing a process in the system.
3. The \_\_init\_\_ method initializes the process attributes such as ID, program counter, memory limit, state, and priority.
4. The set\_state method sets the state of the process.

### **Process Execution:**

def process\_task(pcb):

print(f"Process {pcb.process\_id} with PID {multiprocessing.current\_process().pid} is executing.")

if random.uniform(0, 1) < 0.3:

time.sleep(random.uniform(0.1, 0.5))

pcb.set\_state("TERMINATED")

print(f"Process {pcb.process\_id} with PID {multiprocessing.current\_process().pid} completed.")

else:

if random.uniform(0, 1) < 0.4:

time.sleep(random.uniform(0.1, 0.5))

pcb.set\_state("READY")

print(f"Process {pcb.process\_id} interrupted. Putting back in the ready queue.")

ready\_queue.put(pcb)

return

else:

time.sleep(random.uniform(0.1, 0.5))

pcb.set\_state("WAITING")

print(f"Process {pcb.process\_id} waiting for I/O. Putting in the waiting queue.")

io\_waiting\_queue.put(pcb)

return

1. This function represents the task executed by a process.
2. It prints a message indicating that a process is executing and then simulates whether the process is completed, interrupted, or waiting for I/O based on random probabilities( for sake of simulation).
3. Some amount of sleep time is added to simulate process execution.

### **Long Term Scheduler:**

def long\_term\_scheduler(job\_pool, ready\_queue):

processes = []

# Retrieve processes from the job pool

while not job\_pool.empty():

processes.append(job\_pool.get())

# Sort processes based on priority in descending order

processes.sort(key=lambda x: x.priority, reverse=True)

for pcb in processes:

pcb.set\_state("READY")

ready\_queue.put(pcb)

# Simulate the scheduler making decisions based on priorities, etc.

time.sleep(0.2)

1. This function represents the long-term scheduler.
2. It retrieves processes from the job pool, sorts them based on priority, sets the state and puts them into the ready queue.
3. It also simulates the scheduler making decisions based on priorities. It prioritizes more important tasks by sorting them based on their priority.

### 

### **Short Term Scheduler:**

def short\_term\_scheduler(ready\_queue):

try:

while True:

if not ready\_queue.empty():

pcb = ready\_queue.get()

pcb.set\_state("RUNNING")

process = multiprocessing.Process(target=process\_task, args=(pcb,))

process.start()

process.join()

except KeyboardInterrupt:

print("Short-term scheduler terminated.")

1. This function represents the short-term scheduler.
2. It continuously checks the ready queue for processes, executes their tasks in separate processes, sets their state and waits for them to complete.

### **Handle Waiting Processes Function:**

# Define a function to handle waiting processes

def handle\_waiting\_processes(io\_waiting\_queue):

while True:

if not io\_waiting\_queue.empty():

pcb = io\_waiting\_queue.get()

pcb.set\_state("READY")

ready\_queue.put(pcb)

print(f"Process {pcb.process\_id} moved from waiting to ready queue.")

1. This function handles processes waiting for I/O.
2. It continuously checks the I/O waiting queue, moves processes back to the ready queue when they are ready, sets the state of process and prints a message indicating the transition.

### **Main Function:**

if \_\_name\_\_== "\_\_main\_\_":

num\_processes = 7

job\_pool = multiprocessing.Queue()

ready\_queue = multiprocessing.Queue()

io\_waiting\_queue = multiprocessing.Queue()

# Enqueue processes into the job pool with varying program counters and memory limits

for i in range(num\_processes):

program\_counter = random.randint(100, 1000)

memory\_limit = random.randint(512, 2048)

pcb = PCB(i, program\_counter, memory\_limit)

job\_pool.put(pcb)

# Start the long-term scheduler in a separate process

long\_term\_scheduler\_process = multiprocessing.Process(target=long\_term\_scheduler, args=(job\_pool, ready\_queue))

long\_term\_scheduler\_process.start()

# Start the short-term scheduler in a separate process

short\_term\_scheduler\_process = multiprocessing.Process(target=short\_term\_scheduler, args=(ready\_queue,))

short\_term\_scheduler\_process.start()

# Start the waiting process handler in a separate process

waiting\_handler\_process = multiprocessing.Process(target=handle\_waiting\_processes, args=(io\_waiting\_queue,))

waiting\_handler\_process.start()

try:

# Wait for the long-term scheduler and short-term scheduler processes to finish

long\_term\_scheduler\_process.join()

short\_term\_scheduler\_process.join()

waiting\_handler\_process.join()

except KeyboardInterrupt:

print("Main process terminated.")

print("Simulation completed.")

1. This is the main block of the program.
2. Initialization:
3. Create multiprocessing queues (job\_pool, ready\_queue, io\_waiting\_queue).
4. Enqueue processes into the job\_pool with random program counters and memory limits.
5. Start Processes: Start three separate processes:
6. long\_term\_scheduler\_process: Sorts and schedules processes from job\_pool to ready\_queue.
7. short\_term\_scheduler\_process: Executes tasks from ready\_queue.
8. waiting\_handler\_process: Manages processes waiting for I/O.
9. Wait for Processes:
10. Wait for all scheduler processes to finish execution (join() method).
11. If interrupted (KeyboardInterrupt), terminate gracefully.
12. This main block orchestrates the initialization, execution, and termination of the multiprocessing components, ensuring proper coordination and management of processes and queues involved in the simulation.