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:
   A. Create multiprocessing queues (job_pool, ready_queue, io_waiting_queue).
   B. Enqueue processes into the job_pool with random program counters and memory limits.
3. Start Processes: Start three separate processes:
   A. long_term_scheduler_process: Sorts and schedules processes from job_pool to ready_queue.
   B. short_term_scheduler_process: Executes tasks from ready_queue.
   C. waiting_handler_process: Manages processes waiting for I/O.
4. Wait for Processes:
   A. Wait for all scheduler processes to finish execution (join() method).
   B. If interrupted (KeyboardInterrupt), terminate gracefully.
5. 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.