Sixth Assignment Report

<Sara Akbarzadeh><401222155>

CPU_Simulator

The class contains a nested static class named Task which implements the Runnable interface. The Task class has two instance variables: processingTime, which is the amount of time it takes to process the task, and ID, which is a string identifier for the task. The run() method of the Task class sleeps for the amount of time specified by the processingTime variable.

The CPU_Simulator class contains a method named startSimulation which takes an ArrayList of Task objects and returns an ArrayList of strings representing the IDs of the tasks that were executed. The startSimulation method uses a while loop to iterate through the list of tasks and selects the task with the shortest processing time. The method then creates a new thread to run the selected task and waits for it to finish using the join() method. Once the task has finished, its ID is added to the list of executed tasks and it is removed from the list of remaining tasks. The while loop continues until all tasks have been executed.

The main method of the CPU_Simulator class creates an arraylist of tasks with random processing times and then executes them using the startSimulation method. The list of executed tasks is then printed to the console.

FindMultiples

The getSum(int n) method of the FindMultiples class computes the sum of all multiples of 3, 5, or 7 in the range [1, n]. The method does this by creating a thread pool with 10 threads, submitting a task for each multiple in the range [1, n], waiting for all tasks to finish, and summing up the unique multiples stored in the Task.multiples set.

The Task class is a Runnable that stores a single integer, num, and adds num to the Task.multiples set if num is a multiple of 3, 5, or 7.

The main method of the class creates an instance of FindMultiples and calls its getSum method with n set to 10, printing the resulting sum to the console.

UseInterrupts

The main method of the UseInterrupts class creates instances of both nested classes, starts them, and checks if each thread runs for longer than 3 seconds. If a thread runs for longer than 3 seconds, the main thread interrupts the running thread.

The SleepThread class extends Thread and has a constructor that takes an integer parameter sleepCounter. The run method of the SleepThread class sleeps for 1 second until the sleepCounter is reduced to zero. The run method also prints the number of sleeps remaining at each iteration. If the thread is interrupted while sleeping, the run method prints a message indicating that the thread has been interrupted and terminates the thread.

The LoopThread class also extends Thread and has a constructor that takes an integer parameter value. The run method of the LoopThread class executes a loop 10 times, decrementing the loop variable i by value at each iteration. If the thread is interrupted at any point during the loop, the run method prints a message indicating that the thread has been interrupted and terminates the thread.

In the main method, the SleepThread and LoopThread are started, and the main thread sleeps for 3 seconds before checking if each thread is still alive. If a thread is still alive after 3 seconds, the main thread interrupts it, causing it to terminate.

Bonus Objectives

I created a class “DualCoreCPU_Simulator” that simulates a dual-core CPU by running two tasks concurrently. The program defines a Task class that implements the Runnable interface, with two attributes: ID, which is a string representing the task ID, and processingTime, which is a long representing the time required to complete the task. The Task class also overrides the run method, which contains the logic for executing the task by calling the Thread.sleep method.

The main method of the program creates an ArrayList of Task objects with random processing times. It then creates an instance of the DualCoreCPU_Simulator class and calls its startSimulation method, passing in the list of tasks. The startSimulation method implements the dual-core CPU simulation logic by repeatedly finding the two tasks with the shortest processing times, starting two threads to run these tasks concurrently, waiting for both threads to finish, and adding the completed tasks to an executedTasks list. The method returns the executedTasks list, which is printed to the console in the main method.