Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

З комп’ютерного практикуму № 3 з дисципліни

«Технології паралельних обчислень»

Виконала студентка ІП-13 Замковий Дмитро Володимирович

(шифр, прізвище, ім'я, по батькові)

Перевірив ст.викл. Дифучин Антон Юрійович\_\_\_\_\_

( прізвище, ім'я, по батькові)

Оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_

Дата оцінювання \_\_\_\_\_\_\_\_\_\_\_\_\_

Київ 2024

# **Лабораторна робота 3**

## **Завдання**

1. Реалізуйте програмний код, даний у лістингу, та протестуйте його при різних значеннях параметрів. Модифікуйте програму, використовуючи методи управління потоками, так, щоб її робота була завжди коректною. Запропонуйте три різних варіанти управління. **30 балів.**
2. Реалізуйте приклад Producer-Consumer application (див. https://docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html ). Модифікуйте масив даних цієї програми, які читаються, у масив чисел заданого розміру (100, 1000 або 5000) та протестуйте програму. Зробіть висновок про правильність роботи програми. **20 балів.**
3. Реалізуйте роботу електронного журналу групи, в якому зберігаються оцінки з однієї дисципліни трьох груп студентів. Кожного тижня лектор і його 3 асистенти виставляють оцінки з дисципліни за 100-бальною шкалою. **40 балів.**
4. Зробіть висновки про використання методів управління потоками в java. **10 балів.**

## **Лістинги програми**

### **Файл task1/AsyncBankTest**

public class AsyncBankTest {  
 public static final int *NACCOUNTS* = 10;  
 public static final int *INITIAL\_BALANCE* = 1\_000\_000;  
  
 public static void main(String[] args) {  
 Bank b = new Bank(*NACCOUNTS*, *INITIAL\_BALANCE*);  
   
 for (int i = 0; i < *NACCOUNTS*; i++){  
 TransferThread t = new TransferThread(b, i, *INITIAL\_BALANCE*);  
 t.setPriority(Thread.*NORM\_PRIORITY* + i % 2);  
 t.start();  
 }  
 }  
}

### **Файл task1/Bank**

import java.util.Arrays;  
import java.util.concurrent.locks.ReentrantLock;  
  
class Bank {  
 public static final int *NTEST* = 1000;  
 private final int[] accounts;  
 private long NTransacts = 0;  
 private final ReentrantLock reentrantLock;  
  
 public Bank(int n, int initialBalance){  
 accounts = new int[n];  
 Arrays.*fill*(accounts, initialBalance);  
 reentrantLock = new ReentrantLock();  
 }  
  
 public synchronized void transfer(int from, int to, int amount) {  
 accounts[from] -= amount;  
 accounts[to] += amount;  
 NTransacts++;  
  
 if (NTransacts % *NTEST* == 0)  
 test();  
 }  
  
// public void transfer(int from, int to, int amount) {  
// synchronized (this) {  
// accounts[from] -= amount;  
// accounts[to] += amount;  
// NTransacts++;  
// }  
//  
// if (NTransacts % NTEST == 0) {  
// test();  
// }  
// }  
  
// public void transfer(int from, int to, int amount) {  
// reentrantLock.lock();  
// try {  
// accounts[from] -= amount;  
// accounts[to] += amount;  
// NTransacts++;  
//  
// if (NTransacts % NTEST == 0) {  
// test();  
// }  
// } finally {  
// reentrantLock.unlock();  
// }  
// }  
  
 public synchronized void test(){  
 int sum = 0;  
  
 for (int account : accounts)  
 sum += account;  
  
 System.*out*.println("Transactions:" + NTransacts + " Sum: " + sum);  
 }  
  
 public int size(){  
 return accounts.length;  
 }  
}

### **Файл task1/TransferThread**

class TransferThread extends Thread {  
 private final Bank bank;  
 private final int fromAccount;  
 private final int maxAmount;  
 private static final int *REPS* = 1000;  
   
 public TransferThread(Bank b, int from, int max){  
 bank = b;  
 fromAccount = from;  
 maxAmount = max;  
 }  
  
 @Override  
 public void run() {  
 while (true) {  
 for (int i = 0; i < *REPS*; i++) {  
 int toAccount = (int) (bank.size() \* Math.*random*());  
 int amount = (int) (maxAmount \* Math.*random*()/*REPS*);  
 bank.transfer(fromAccount, toAccount, amount);  
 }  
 }  
 }  
}

### **Файл task2/Consumer**

public class Consumer implements Runnable {  
 private final Drop drop;  
 public Consumer(Drop drop) {  
 this.drop = drop;  
 }  
  
 public void run() {  
 for (int message = drop.take(); message != -1; message = drop.take()) {  
 System.*out*.format("MESSAGE RECEIVED: %d%n", message);  
  
 try {  
 Thread.*sleep*(50);  
 } catch (InterruptedException ignored) {}  
 }  
  
 System.*out*.println("DONE");  
 }  
}

### **Файл task2/Drop**

public class Drop {  
 private int message;  
 private boolean empty = true;  
 public synchronized int take() {  
 while (empty) {  
 try {  
 wait();  
 } catch (InterruptedException ignored) {}  
 }  
  
 empty = true;  
 notifyAll();  
 return message;  
 }  
  
 public synchronized void put(int message) {  
 while (!empty) {  
 try {  
 wait();  
 } catch (InterruptedException ignored) {}  
 }  
  
 empty = false;  
 this.message = message;  
 notifyAll();  
 }  
}

### **Файл task2/Main**

public class Main {  
 public static void main(String[] args) {  
 Drop drop = new Drop();  
  
 (new Thread(new Producer(drop))).start();  
 (new Thread(new Consumer(drop))).start();  
 }  
}

### **Файл task2/Producer**

public class Producer implements Runnable {  
 private final int SIZE = 100;  
 private Drop drop;  
  
 public Producer(Drop drop) {  
 this.drop = drop;  
 }  
  
 public void run() {  
 int[] info = new int[SIZE];  
  
 for (int i = 0; i < SIZE; i++) {  
 info[i] = i + 1;  
 }  
  
 for (int i = 0; i < SIZE; i++) {  
 drop.put(info[i]);  
 try {  
 Thread.*sleep*(50);  
 } catch (InterruptedException ignored) {}  
 }  
  
 drop.put(-1);  
 }  
}

### **Файл task3/Group**

import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
  
public class Group {  
 private final HashMap<Integer, List<Integer>> studentList;  
 private final int MAX\_STUDENTS;  
  
 public Group(int maxStudents) {  
 this.MAX\_STUDENTS = maxStudents;  
 studentList = new HashMap<>();  
 for (int i = 0; i < MAX\_STUDENTS; i++) {  
 List<Integer> marks = new ArrayList<>();  
 studentList.put(i, marks);  
 }  
 }  
  
 public synchronized void addMark(Integer student, int mark) {  
 List<Integer> studentMarks = studentList.get(student);  
 studentMarks.add(mark);  
 }  
  
 public List<Integer> getStudents() {  
 return studentList.keySet().stream().toList();  
 }  
  
 public List<Integer> getMarks(Integer student) {  
 return studentList.get(student);  
 }  
}

### **Файл task3/Journal**

import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
public class Journal {  
 private final HashMap<String, Group> groups;  
  
 public Journal(HashMap<String, Integer> students) {  
 this.groups = new HashMap<>();  
   
 for (Map.Entry<String, Integer> entry : students.entrySet()) {  
 String key = entry.getKey();  
 Integer value = entry.getValue();  
 this.groups.put(key, new Group(value));  
 }  
 }  
  
 public void gradeStudent(String groupCode, Integer student, int mark) {  
 Group group = groups.get(groupCode);  
 group.addMark(student, mark);  
 }  
  
 public void printJournal() {  
 for (Map.Entry<String, Group> entry : groups.entrySet()) {  
 System.*out*.println("\nGroup " + entry.getKey() + ":");  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*");  
 for (Integer student : entry.getValue().getStudents()) {  
 System.*out*.println("Студент " + student + ": " + entry.getValue().getMarks(student));  
 }  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*");  
 }  
 }  
  
 public List<Integer> getStudents(String groupCode) {  
 return groups.get(groupCode).getStudents();  
 }  
}

### **Файл task3/Main**

import java.util.HashMap;  
import java.util.List;  
  
public class Main {  
 public static void main(String[] args) {  
 Journal journal = new Journal(new HashMap<>() {{  
 put("G-1", 22);  
 put("G-2", 27);  
 put("G-3", 18);  
 }});  
  
 TeacherThread teacherThread1 = new TeacherThread(journal, List.*of*("G-1", "G-2", "G-3"), "Lecturer");  
 TeacherThread teacherThread2 = new TeacherThread(journal, List.*of*("G-1"), "Assistant 1");  
 TeacherThread teacherThread3 = new TeacherThread(journal, List.*of*("G-2"), "Assistant 2");  
 TeacherThread teacherThread4 = new TeacherThread(journal, List.*of*("G-3"), "Assistant 3");  
  
 Thread thread1 = new Thread(teacherThread1);  
 Thread thread2 = new Thread(teacherThread2);  
 Thread thread3 = new Thread(teacherThread3);  
 Thread thread4 = new Thread(teacherThread4);  
  
 thread1.setPriority(Thread.*MAX\_PRIORITY*);  
  
 thread1.start();  
 thread2.start();  
 thread3.start();  
 thread4.start();  
  
 journal.printJournal();  
 }  
}

### **Файл task3/TeacherThread**

import java.util.List;  
  
public class TeacherThread implements Runnable {  
 private final String name;  
 private final Journal journal;  
 private final List<String> groupCode;  
  
 public TeacherThread(Journal journal, List<String> groupCode, String name) {  
 this.name = name;  
 this.journal = journal;  
 this.groupCode = groupCode;  
 }  
  
 @Override  
 public void run() {  
 for (int i = 0; i < 5; i++) {  
 for (String code : groupCode) {  
 for (Integer student : journal.getStudents(code)) {  
 journal.gradeStudent(code, student, (int) (Math.*random*() \* 100));  
 }  
 }  
 }  
 }  
}

## **Скріншоти**

Завдання 1

Зображення, що містить знімок екрана, текст

Автоматично згенерований опис

Завдання 2

Зображення, що містить знімок екрана, текст, комп’ютер

Автоматично згенерований опис

Завдання 3

Зображення, що містить текст, знімок екрана

Автоматично згенерований опис

## **Висновки**

В ході даної лабораторної роботи ми дослідили способи управління потоками і винесли наступні висновки: багатопоточність дозволяє ефективніше використовувати багатоядерні процесори, розділяючи задачі між ядрами; потрібно правильно синхроніззувати код, оскільки різні потоки можуть використовувати одну і ту ж пам’ять одночасно і через це виникають помилки.