**МИНИСТЕРСТВО**

**ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ  
ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ  
 УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

по дисциплине: «**Визуальные средства разработки программных приложений»**

на тему: «Разработка многопоточных приложений»

Выполнил: студент гр. ИТП-31

Расшивалов Н.И.  
 Принял: ассистент

Михалевич В.Г.

Гомель 2022

**Цель работы:** изучить создание многопоточных приложений с использованием языка программирования *Java*.

Разработать приложение, в котором выполняется следующий алгоритм: два параллельных потока циклически выполняют вызов функций. Каждый поток использкет свою функцию. Результаты работы каждый поток помещает в общую очередь конечной длины. Третий поток забирает из очереди результаты работы функций и отображает их на экране в происвольной форме.

Схема взаимодействия потоков приведена на рисунке 1.

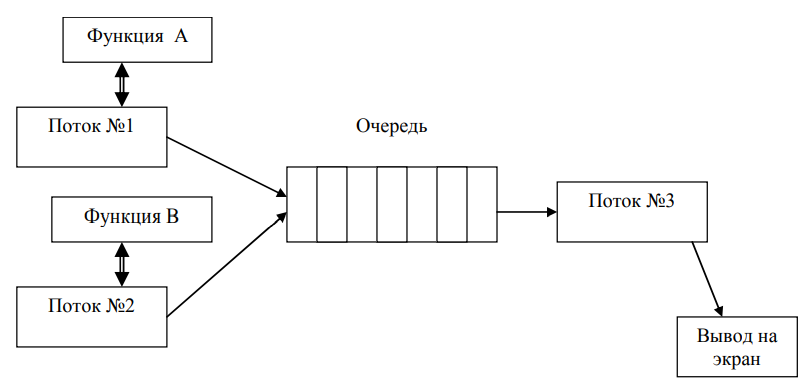


Рисунок 1 – Схема взаимодействия потоков

Выбранные функции: сложение и умножение чисел в восьмеричной системе исчисления.

Результат работы программы представлен на рисунке 2, где видно параллельное изменение очереди результатов.

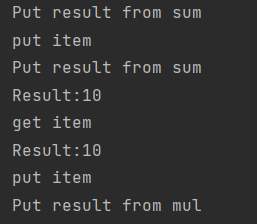


Рисунок 2 – Результат работы программы

Листинг программы представлен в приложении А.

**Вывод:** в ходе выполнения лабораторной работы, изучены применение потоков и разработка многопоточных приложений в *Java*.

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

package com.company;  
  
import Functions.MulFunction;  
import Functions.SumFunction;  
import Queues.SyncQueue;  
import Threads.ConsumerThread;  
import Threads.FunctionThread;  
  
public class Main {  
  
 public static void main(String[] args) {  
 var queue = new SyncQueue<Integer>(6);  
 var consumer = new ConsumerThread(queue);  
 var mulFunction = new FunctionThread<Integer>(new MulFunction(2,4), queue);  
 var sumFunction = new FunctionThread<Integer>(new SumFunction(7,1), queue);  
 var consumerThread = new Thread(consumer,"consumer");  
 var sumFunctionThread = new Thread(sumFunction, "sumFunction");  
 var mulFunctionThread = new Thread(mulFunction, "subFunction");  
 sumFunctionThread.start();  
 mulFunctionThread.start();  
 consumerThread.start();  
 }  
}

package Functions;  
  
import Queues.SyncQueue;  
  
import javax.management.InvalidApplicationException;  
import java.util.Queue;  
import java.util.concurrent.SynchronousQueue;  
  
public abstract class Function<T> {  
 protected T[] \_values;  
 protected int \_current;  
 private String \_name;  
  
 public String getName(){  
 return \_name;  
 }  
  
 public Function(String name, T[] values){  
 \_values = values;  
 \_name = name;  
 }  
  
 public boolean hasNext(){  
 if(\_current < \_values.length){  
 return true;  
 }  
  
 return false;  
 }  
  
 public abstract T execute() throws InvalidApplicationException;  
}

package Functions;  
  
import javax.management.InvalidApplicationException;  
  
public class MulFunction extends Function<Integer> {  
  
 public MulFunction(Integer... values){  
 super("sub", values);  
 \_values = values;  
 }  
  
 public Integer execute() throws InvalidApplicationException {  
 if(\_current < \_values.length) {  
 var result = 1;  
 for (var i = 0; i < \_values.length ; i++) {  
 result \*= Integer.parseInt(\_values[i].toString(), 8);  
 }  
 \_current += 1;  
 result = Integer.parseInt(Integer.toOctalString(result));  
 return result;  
 }  
 throw new InvalidApplicationException("There is no any input");  
 }  
}

package Functions;  
  
import javax.management.InvalidApplicationException;  
  
public class SumFunction extends Function<Integer> {  
  
 public SumFunction(Integer... values){  
 super("sum", values);  
 \_values = values;  
 }  
  
 public Integer execute() throws InvalidApplicationException {  
 if(\_current < \_values.length) {  
 var result = 0;  
 for (var i = 0; i < \_values.length ; i++) {  
 result += Integer.parseInt(\_values[i].toString(), 8);  
 }  
 \_current += 1;  
 result = Integer.parseInt(Integer.toOctalString(result));  
 return result;  
 }  
 throw new InvalidApplicationException("There is no any input");  
 }  
}

package Queues;  
  
import java.util.ArrayDeque;  
import java.util.Queue;  
  
public class SyncQueue<T> {  
 private int \_maxLength;  
 private Queue<T> \_queue = new ArrayDeque<T>();  
  
 public SyncQueue(int maxLength){  
 \_maxLength = maxLength;  
 }  
  
 public T get(){  
 while(true) {  
 synchronized (\_queue) {  
 if (\_queue.size() != 0) {  
 System.out.println("get item");  
 return \_queue.poll();  
 }  
 }  
 }  
 }  
  
 public synchronized boolean hasNext(){  
 if(\_queue.size() > 0){  
 return true;  
 }  
  
 return false;  
 }  
  
  
 public void put(T item){  
 while(true) {  
 synchronized (\_queue) {  
 if (\_queue.size() < \_maxLength) {  
 System.out.println("put item");  
 \_queue.add(item);  
 return;  
 }  
 }  
 }  
 }  
}

package Threads;  
  
import Functions.Function;  
import Queues.SyncQueue;  
  
public class ConsumerThread implements Runnable {  
 private SyncQueue<Integer> \_queue;  
 public ConsumerThread(SyncQueue<Integer> queue){  
 \_queue = queue;  
 }  
  
 @Override  
 public void run() {  
 while(true) {  
 System.out.println("Result:" + \_queue.get());  
 }  
 }  
}

package Threads;  
  
import Functions.Function;  
import Queues.SyncQueue;  
  
import javax.management.InvalidApplicationException;  
  
public class FunctionThread<T> implements Runnable {  
 public Function<T> \_function;  
 private SyncQueue<T> \_queue;  
 public FunctionThread(Function<T> function, SyncQueue<T> queue){  
 \_function = function;  
 \_queue = queue;  
 }  
  
  
 @Override  
 public void run() {  
 while (\_function.hasNext()) {  
 T result = null;  
 try {  
 result = \_function.execute();  
 } catch (InvalidApplicationException e) {  
 e.printStackTrace();  
 }  
 \_queue.put(result);  
 System.out.println("Put result from " + \_function.getName());  
 }  
 }  
}