Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

К У Р С О В А Я Р А Б О Т А

Работа с потоками в ОС Linux.

Тема: “Заправочная станция с несколькими бензоколонками”

по дисциплине «Операционные системы»

Выполнили
студенты гр. 5130901/10101 М.В. Писарик
Д.А.Тучков

Руководитель
Старший преподаватель З.В. Куляшова

«___» _____ 2023 г.

Санкт-Петербург

2023

Оглавление

1. Введение. Постановка цели исследования и описание задач для достижения поставленной цели.

Задачи очереди:

Параллельное программирование позволяет разделить программу на несколько независимых частей. Довольно часто бывает необходимо превратить программу в несколько отдельных, самостоятельно выполняющихся подзадач. Каждая из этих самостоятельных подзадач называется потоком (thread). Поток – это выполняемая параллельно в рамках процесса последовательность команд программы.

Для демонстрации эффективности использования многопоточного программирования в некоторой ситуации была решена задача - заправочная станция с несколькими бензоколонками. Одновременная заправка нескольких машин и оплата топлива. Для ее программной реализации был использован язык программирования Java, а для реализации многопоточности был использован Class Thread.

2. Подробное описание задачи, которая будет реализована в приложении.

Необходимо разработать модель заправки. Она должна обеспечивать одновременную заправку машин в рамках очереди, а также программа должна реализовывать оплату бензина.

3. Описание базового алгоритма, который планируется в многопоточной реализации.

Многопоточность на уровне бензоколонки (GasPump):

- Для каждой бензоколонки создается отдельный поток, который выполняет метод startRefueling.
- В методе startRefueling используется бесконечный цикл, в котором проверяется, свободна ли текущая колонка и есть ли машины в очереди.
- Если колонка свободна и в очереди есть машина, она начинает заправку.
- Заправка симулируется с использованием временной задержки, представляющей время, необходимое для заправки автомобиля.

```
class GasPump {
    5 usages
    String fuelType;
    6 usages
    Queue<Car> carQueue;
    4 usages
    double pricePerLiter;
    6 usages
    double litersAvailable;
    6 usages
    boolean isFree;
    4 usages
    Thread pumpThread;
```

```java
void startRefueling() {
    while (true) {
        try {
            TimeUnit.MILLISECONDS.sleep( timeout: 100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        synchronized (this) {
            if (isFree && !carQueue.isEmpty()) {
                Car car = carQueue.poll();
                if (check(car)) {
                    a++;
                    // System.out.println(a);
                    a--;
                    isFree = false;
                    try {
                        this.litersAvailable -= car.litersNeeded;
                        TimeUnit.MILLISECONDS.sleep((int) (car.litersNeeded * 500));
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    if (car.fuelType.equals("накачка")) {
                        System.out.printf("Колеса автомобиля %d успешно накачаны %n", car.id);
                    } else {

                        System.out.printf("Автомобиль %d успешно заправлен%n", car.id);
                        decrementI();

                    }
                    isFree = true;
                }
            }
        }
    }
```

Использование многопоточности на уровне главного потока (main):

- При создании объекта **GasStation**, запускаются потоки для каждой бензоколонки с помощью **pumpThread.start()**.
- Главный поток ожидает ввода данных от пользователя в бесконечном цикле.

```
GasStation() {
    this.pumps = new HashMap<>();
    this.carQueues = new HashMap<>();
    // add_pump("92", 50.56, 1000000000);

    for (List<GasPump> pump_list : pumps.values()) { // паралельная работа бензоколонки
        for (GasPump pump : pump_list) {
            pump.pumpThread = new Thread(pump::startRefueling);
            pump.pumpThread.start();
        }
    }
}
```

TimeUnit.MILLISECONDS.sleep() - используется для введения задержек в выполнении потоков. Для симуляции времени заправки или накачки

```
TimeUnit.MILLISECONDS.sleep((int) (car.litersNeeded * 500));
```

Принуып работы:

```
pump.pumpThread = new Thread(pump::startRefueling);
pump.pumpThread.start();
```

- Создается новый объект класса **Thread**, и ему в конструктор передается ссылка на метод **startRefueling** объекта **pump**.
- **pump::startRefueling** - это ссылка на метод (method reference). Она указывает на метод **startRefueling** объекта **pump**.
- Таким образом, создается новый поток, который будет выполнять код метода **startRefueling** объекта **pump**.

- Запускается созданный поток, вызывая метод **start()** на объекте **pumpThread**.
- Этот метод **start()** инициирует выполнение кода в методе **run()** для данного потока, который внутри себя вызывает **startRefueling.**

(start() создает новый поток и вызывает **run()** внутри этого потока.)

4. Описание основных процедур (основной акцент на описание частей программы, связанных с реализацией алгоритма по потокам) с примерами кода реализации.

В рамках программы существует множество процедур. Первостепенно программа выводит в консоль информацию о типах топлива, имеющихся на станции, их стоимости, количеству, а также информацию об услуги накачки шин.

```
Заправочная станция
Тип топлива: 100;  цена за Литр: 69.54 руб;  Имеется литров на станции: 100.0
Тип топлива: DF;  цена за Литр: 62.55 руб;  Имеется литров на станции: 100.0
Тип топлива: 92;  цена за Литр: 50.56 руб;  Имеется литров на станции: 100.0
Тип топлива: 95;  цена за Литр: 55.26 руб;  Имеется литров на станции: 100.0
Накачка шин бесплатно в любых объемах
```

Далее программа принимает на вход строку формата: <код топлива>_<предаваемая сумма денег у.е.>_<количество литров бензина>
Или формата: <накачка>_<объём воздуха, необходимый для накачки колес авто.>

main method:

main(String[] args): Основной метод программы, который инициализирует заправочную станцию, сканер и обрабатывает пользовательский ввод в бесконечном цикле.

```java
public class main {

    public static void main(String[] args) {

        GasStation station = new GasStation();
        //oneThread(station);
         multiThread(station);

        Map<String, List<GasPump>> pumps = station.get_pumps();
        // Ваш существующий код обработки строки
        System.out.println("Заправочная станция");
        for (List<GasPump> pump_list : pumps.values()) { // паралельная работа бензоколонки
            for (GasPump pump : pump_list) {
                if (pump.fuelType.equals("накачка"))
                    System.out.println("Накачка шин бесплатно в любых объемах");
                else
                    System.out.println("Тип топлива: " + pump.fuelType + ";  цена за Литр: " + pump.pricePerLiter
                            + " руб" + ";  Имеется литров на станции: " + pump.litersAvailable);
            }
        }

         try (BufferedReader reader = new BufferedReader(new
         FileReader( fileName: "C:\\Users\\Dima\\Desktop\\os\\car.txt"))) {
         String line;
        while ((line = reader.readLine()) != null) {
        processInputLine(line, station);
         }
         } catch (IOException e) {
         e.printStackTrace();
         }

        // Принятие ввода с консоли
         Scanner consoleScanner = new Scanner(System.in);
         System.out.println("Введите данные (для завершения введите 'exit'):");

         while (true) {
        String inputLine = consoleScanner.nextLine();
         if (inputLine.equals("exit")) {
         break;
```

Обработка неверного формата ввода данных:

```
92 100
Неверные входные данные! Правильный вариант: - "<тип топлива> <деньги> <необходимые литры>"
92 abc 20
Неправильный формат ввода данных.
100 -5 11
Неправильный формат ввода данных.
```

Car class:

Car представляет объект автомобиля и содержит информацию о типе топлива, деньгах, необходимых литрах и идентификаторе.

```java
class Car {
    9 usages
    String fuelType;
    3 usages
    double money;
    5 usages
    double litersNeeded;

    9 usages
    int id;
    1 usage
    static int nextId = 1;

    4 usages
    Car(String fuelType, double money, double litersNeeded) {
        this.fuelType = fuelType;
        this.money = money;
        this.litersNeeded = litersNeeded;
        this.id = nextId++;
    }
}
```

GasPump class:

- **addCarToQueue(Car car)**: Метод добавляет автомобиль в очередь для заправки на данном бензонасосе.
- **check(Car car)**: Проверяет, достаточно ли топлива на насосе, достаточно ли денег у автомобиля и начинает процесс заправки, выводя соответствующие сообщения.
- **startRefueling()**: Метод, работающий бесконечно, моделирует процесс заправки при наличии свободного насоса и машин в очереди.

```
class GasPump {
    5 usages
    String fuelType;
    6 usages
    Queue<Car> carQueue;
    4 usages
    double pricePerLiter;
    6 usages
    double litersAvailable;
    6 usages
    boolean isFree;
    4 usages
    Thread pumpThread;

    1 usage
    GasPump(String fuelType, double pricePerLiter, double litersAvailable) {
        this.fuelType = fuelType;
        this.pricePerLiter = pricePerLiter;
        this.litersAvailable = litersAvailable;
        this.isFree = true;
        this.carQueue = new LinkedList<>();
    }
  boolean check(Car car) {

      double cost = car.litersNeeded * pricePerLiter;
      if (this.litersAvailable < car.litersNeeded) {
          System.out.printf("На станции недостаточно топлива %s  для Car %s. Имеется %s литров. %
                  car.fuelType, car.id, this.litersAvailable);
          return false;
      } else if (car.money >= cost) {
          double change = car.money - cost;
          if (!car.fuelType.equals("накачка"))
              System.out.printf("Автомобиль %d оплатил заправку, сдача = %.2f руб%n", car.id, char
          System.out.printf("Автомобиль %d начал заправку %n", car.id, Thread.currentThread().geti
          // (№потока: %d)
          return true;
      } else {
          System.out.printf("У автомобиля %d не хватает денег на заправку.%n", car.id);
          return false;
      }
  }
```

В случае, если достаточно средств на заправку:

```
92 10000 10
Автомобиль 9 прибывает на станцию за 92 топливом.
Автомобиль 9 оплатил заправку, сдача = 9494,40 руб
```

В случае недостатка средств:

```
92 1 10
Автомобиль 8 прибывает на станцию за 92 топливом.
У автомобиля 8 не хватает денег на заправку.
```

В случае нехватки топлива на станции:

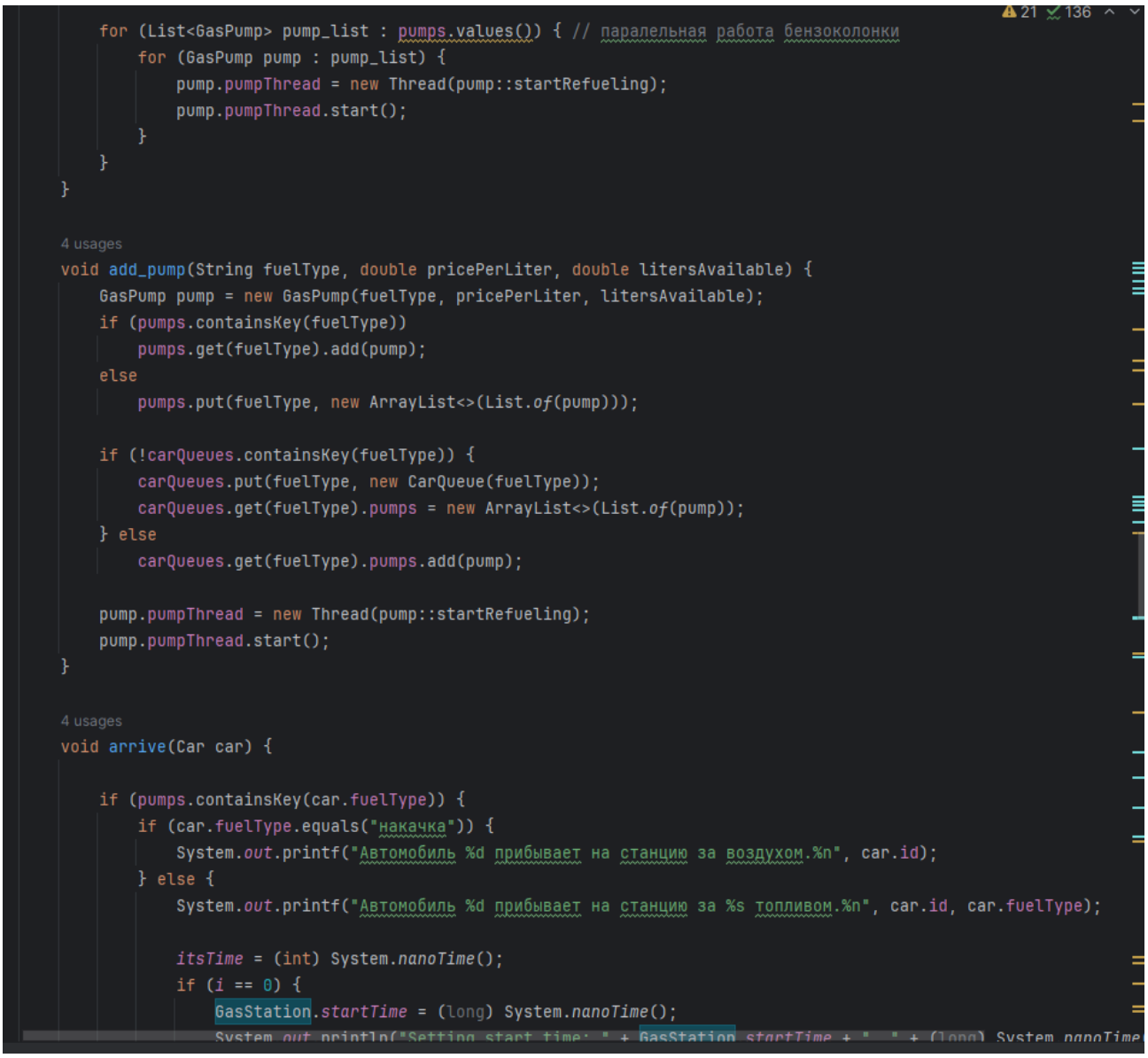
```
92 10000 100
Автомобиль 10 прибывает на станцию за 92 топливом.
На станции недостаточно топлива 92  для Car 10. Имеется 38.0 литров.
```

```java
void startRefueling() {
    while (true) {
        try {
            TimeUnit.MILLISECONDS.sleep( timeout: 100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        synchronized (this) {
            if (isFree && !carQueue.isEmpty()) {
                Car car = carQueue.poll();
                if (check(car)) {
                    a++;
                    // System.out.println(a);
                    a--;
                    isFree = false;
                    try {
                        this.litersAvailable -= car.litersNeeded;
                        TimeUnit.MILLISECONDS.sleep((int) (car.litersNeeded * 500));
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    if (car.fuelType.equals("накачка")) {
                        System.out.printf("Колеса автомобиля %d успешно накачаны %n", car.id);
                    } else {

                        System.out.printf("Автомобиль %d успешно заправлен%n", car.id);
                        decrementI();

                    }
                    isFree = true;
                }
            }
```

GasStation class:

- **arrive(Car car)**: Принимает автомобиль и направляет его в соответствующую очередь насоса на основе типа топлива, выводя соответствующие сообщения.

```
class GasStation {

    3 usages
    static long startTime;
    1 usage
    static long itsTime;
    4 usages
    static volatile int i;
    4 usages
    static Lock lock = new ReentrantLock();
    7 usages
    private final Map<String, List<GasPump>> pumps;
    6 usages
    private final Map<String, CarQueue> carQueues;

    3 usages
    public Map<String, List<GasPump>> get_pumps() { return pumps; }

    1 usage
    public synchronized void incrementI() {

     //   System.out.println("In I " + GasStation.i);

        GasStation.lock.lock();
        try {
            GasStation.i++;
        } finally {
            GasStation.lock.unlock();
        }
    }

    1 usage
    GasStation() {
        this.pumps = new HashMap<>();
        this.carQueues = new HashMap<>();
        // add_pump("92", 50.56, 1000000000);
```



```
    for (List<GasPump> pump_list : pumps.values()) { // параллельная работа бензоколонки
        for (GasPump pump : pump_list) {
            pump.pumpThread = new Thread(pump::startRefueling);
            pump.pumpThread.start();
        }
    }
}

4 usages
void add_pump(String fuelType, double pricePerLiter, double litersAvailable) {
    GasPump pump = new GasPump(fuelType, pricePerLiter, litersAvailable);
    if (pumps.containsKey(fuelType))
        pumps.get(fuelType).add(pump);
    else
        pumps.put(fuelType, new ArrayList<>(List.of(pump)));

    if (!carQueues.containsKey(fuelType)) {
        carQueues.put(fuelType, new CarQueue(fuelType));
        carQueues.get(fuelType).pumps = new ArrayList<>(List.of(pump));
    } else
        carQueues.get(fuelType).pumps.add(pump);

    pump.pumpThread = new Thread(pump::startRefueling);
    pump.pumpThread.start();
}

4 usages
void arrive(Car car) {

    if (pumps.containsKey(car.fuelType)) {
        if (car.fuelType.equals("накачка")) {
            System.out.printf("Автомобиль %d прибывает на станцию за воздухом.%n", car.id);
        } else {
            System.out.printf("Автомобиль %d прибывает на станцию за %s топливом.%n", car.id, car.fuelType);

            itsTime = (int) System.nanoTime();
            if (i == 0) {
                GasStation.startTime = (long) System.nanoTime();
                System.out.println("Setting start time: " + GasStation.startTime + " " + (long) System.nanoTime
```

5. Результаты работы приложения. Сравнение работы многопоточного приложения с однопоточной реализацией.

Введя строку “100 1000 10” мы передаем программе информацию о том, что на АЗС поступил автомобиль, его следует заправить бензином с октановым числом 100, водитель авто передал сотруднику заправки 1000 у. е. и необходимо заправить транспорт 10ю литрами бензина.

Получив эту строку, программа передает следующую информацию:

```
100 1000 10
Автомобиль 1 прибывает на станцию за 100 топливом.
Автомобиль 1 оплатил заправку, сдача = 304,60 руб
Автомобиль 1 начал заправку
```

Автомобиль 1 прибывает на станцию, необходим 100-й бензин, кассир вернул сдачу 304,60 руб, Сотрудник АЗС начал заправку авто.

После этого на колонку приехало еще 2 авто, и начали заправку.

```
95 10000 30
Автомобиль 2 прибывает на станцию за 95 топливом.
Автомобиль 2 оплатил заправку, сдача = 8342,20 руб
Автомобиль 2 начал заправку
100 100039 50
Автомобиль 3 прибывает на станцию за 100 топливом.
Автомобиль 3 оплатил заправку, сдача = 96562,00 руб
Автомобиль 3 начал заправку
```

По прошествии нескольких секунд программа выдает сообщение, что вначале заправлен 2ой автомобиль:

```
Автомобиль 3 прибывает на станцию за 100 топливом.
Автомобиль 3 оплатил заправку, сдача = 96562,00 руб
Автомобиль 3 начал заправку
Автомобиль 2 успешно заправлен
```

Потом и 3ий:

```
Автомобиль 3 начал заправку
Автомобиль 2 успешно заправлен
Автомобиль 3 успешно заправлен
```

Как уже было сказано, программа реализует по одному потоку на колонку. Каждой из колонок соответствует определенный тип бензина. В предыдущем примере авто не стояли в очереди и их заправка была параллельной. Процесс заправки завершился практически одновременно. При этом, если мы будем

заправлять автомобили топливом одного вида, то процесс заправки будет реализован одним потоком в рамках очереди. Пример приведен на скриншоте ниже:

```
92 10000 30
Автомобиль 4 прибывает на станцию за 92 топливом.
Автомобиль 4 оплатил заправку, сдача = 8483,20 руб
Автомобиль 4 начал заправку
92 10000 30
Автомобиль 5 прибывает на станцию за 92 топливом.
92 10000 30
Автомобиль 6 прибывает на станцию за 92 топливом.
92 10000 30
Автомобиль 7 прибывает на станцию за 92 топливом.
Автомобиль 4 успешно заправлен
Автомобиль 5 оплатил заправку, сдача = 8483,20 руб
Автомобиль 5 начал заправку
Автомобиль 5 успешно заправлен
Автомобиль 6 оплатил заправку, сдача = 8483,20 руб
Автомобиль 6 начал заправку
Автомобиль 6 успешно заправлен
На станции недостаточно топлива 92  для Car 7. Имеется 10.0 литров.
```

На одну колонку на АЗС прибывает сразу 4 авто, и станция обслуживает клиентов в порядке очереди. Причем, оплату сотрудник принимает у посетителя и начинает заправку только после того, как заправил предыдущую машину. Также, можно увидеть, что клиенты в сумме приобрели 90 литров топлива 92 октанового числа, и станция не может продать посетителю под номером 7 30 литров так, как осталось всего 10.

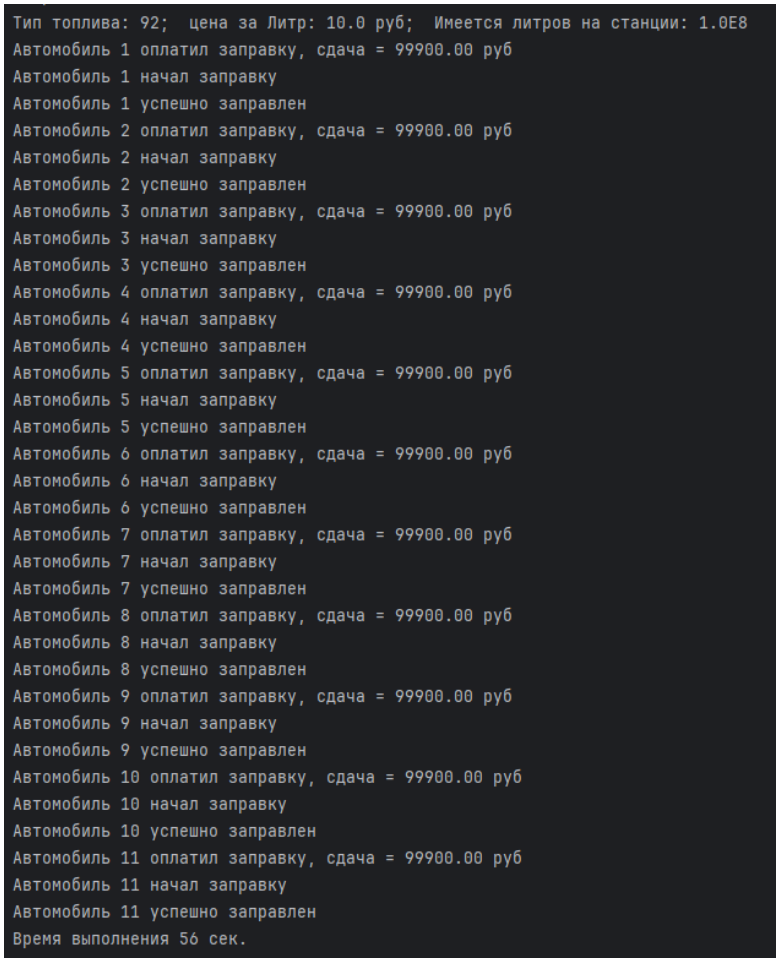
Так же очередь реализована для накачки шин:

```
накачка 100
Автомобиль 1 прибывает на станцию за воздухом.
Автомобиль 1 начал заправку
накачка 10
Автомобиль 2 прибывает на станцию за воздухом.
накачка 30
Автомобиль 3 прибывает на станцию за воздухом.
накачка 2
Автомобиль 4 прибывает на станцию за воздухом.
Колеса автомобиля 1 успешно накачаны
Автомобиль 2 начал заправку
Колеса автомобиля 2 успешно накачаны
Автомобиль 3 начал заправку
```

На скриншоте видно, что авто попадают в одну очередь на накачку шин.

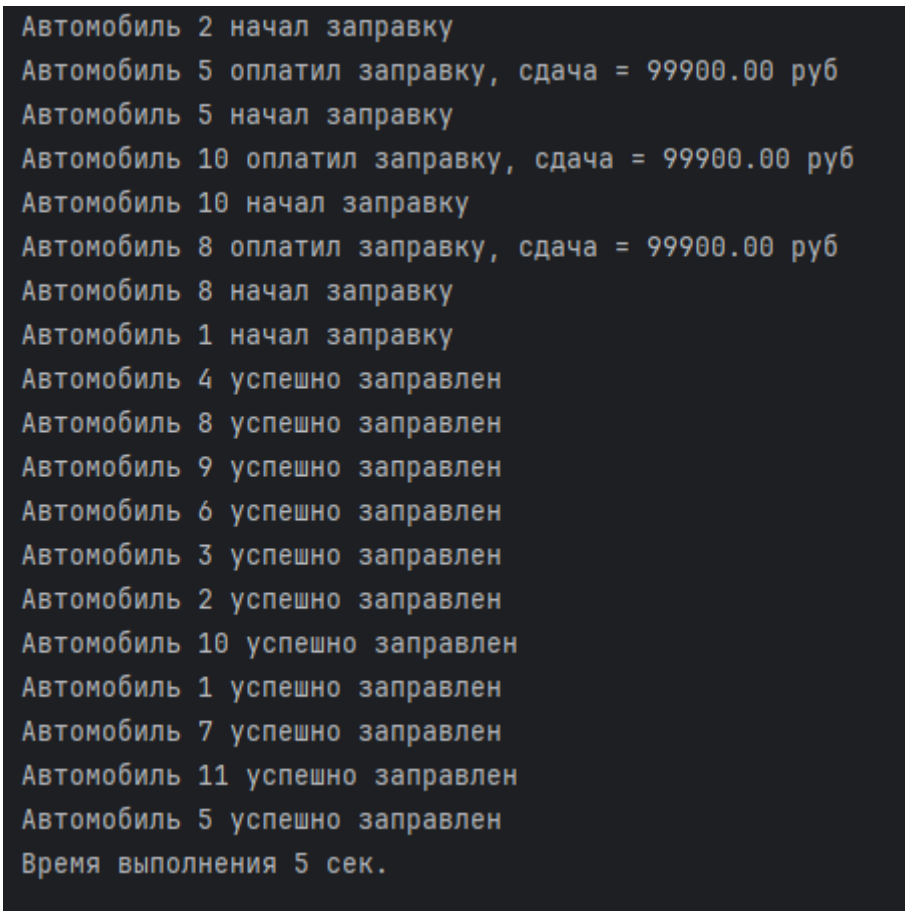
Временные характеристики:

```
Автомобиль 1 прибывает на станцию за 92 топливом.
Setting start time: 1271037097754900  1271037097755500
Автомобиль 2 прибывает на станцию за 92 топливом.
Автомобиль 3 прибывает на станцию за 92 топливом.
Автомобиль 4 прибывает на станцию за 92 топливом.
Автомобиль 5 прибывает на станцию за 92 топливом.
Автомобиль 6 прибывает на станцию за 92 топливом.
Автомобиль 7 прибывает на станцию за 92 топливом.
Автомобиль 8 прибывает на станцию за 92 топливом.
Автомобиль 9 прибывает на станцию за 92 топливом.
Автомобиль 10 прибывает на станцию за 92 топливом.
Автомобиль 11 прибывает на станцию за 92 топливом.
Заправочная станция
```



Время заправки 11 машин в очереди на 1 бензоколонку

```
Автомобиль 1 прибывает на станцию за 92 топливом.
Setting start time: 1271289415551900  1271289415552600
Автомобиль 2 прибывает на станцию за 92 топливом.
Автомобиль 3 прибывает на станцию за 92 топливом.
Автомобиль 4 прибывает на станцию за 92 топливом.
Автомобиль 5 прибывает на станцию за 92 топливом.
Автомобиль 6 прибывает на станцию за 92 топливом.
Автомобиль 7 прибывает на станцию за 92 топливом.
Автомобиль 8 прибывает на станцию за 92 топливом.
Автомобиль 9 прибывает на станцию за 92 топливом.
Автомобиль 10 прибывает на станцию за 92 топливом.
Автомобиль 11 прибывает на станцию за 92 топливом.
Заправочная станция
Тип топлива: 92;  цена за Литр: 10.0 руб;  Имеется литров на станции: 1.0E8
Тип топлива: 92;  цена за Литр: 10.0 руб;  Имеется литров на станции: 1.0E8
Тип топлива: 92;  цена за Литр: 10.0 руб;  Имеется литров на станции: 1.0E8
Тип топлива: 92;  цена за Литр: 10.0 руб;  Имеется литров на станции: 1.0E8
Тип топлива: 92;  цена за Литр: 10.0 руб;  Имеется литров на станции: 1.0E8
Тип топлива: 92;  цена за Литр: 10.0 руб;  Имеется литров на станции: 1.0E8
Тип топлива: 92;  цена за Литр: 10.0 руб;  Имеется литров на станции: 1.0E8
Тип топлива: 92;  цена за Литр: 10.0 руб;  Имеется литров на станции: 1.0E8
Тип топлива: 92;  цена за Литр: 10.0 руб;  Имеется литров на станции: 1.0E8
Тип топлива: 92;  цена за Литр: 10.0 руб;  Имеется литров на станции: 1.0E8
Тип топлива: 92;  цена за Литр: 10.0 руб;  Имеется литров на станции: 1.0E8
Автомобиль 1 оплатил заправку, сдача = 99900.00 руб
Автомобиль 9 оплатил заправку, сдача = 99900.00 руб
Автомобиль 9 начал заправку
Автомобиль 11 оплатил заправку, сдача = 99900.00 руб
Автомобиль 11 начал заправку
Автомобиль 3 оплатил заправку, сдача = 99900.00 руб
Автомобиль 3 начал заправку
Автомобиль 4 оплатил заправку, сдача = 99900.00 руб
Автомобиль 4 начал заправку
Автомобиль 6 оплатил заправку, сдача = 99900.00 руб
Автомобиль 6 начал заправку
Автомобиль 7 оплатил заправку, сдача = 99900.00 руб
Автомобиль 7 начал заправку
Автомобиль 2 оплатил заправку, сдача = 99900.00 руб
```

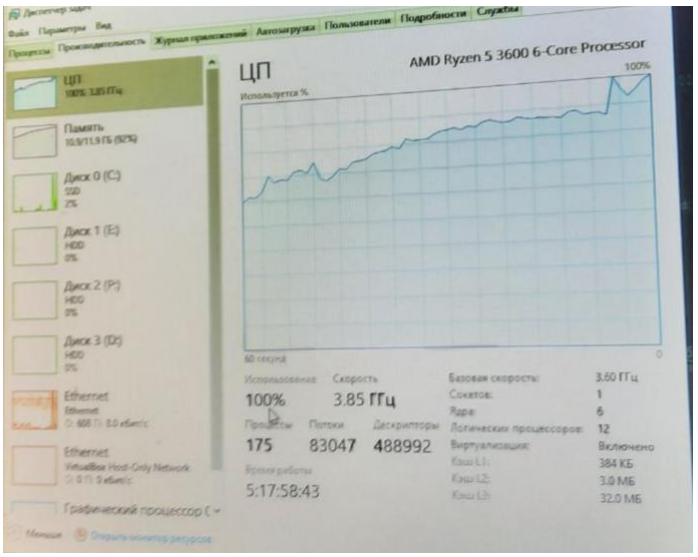


Время заправки 11 автомобилей на 11 бензоколонках

Как видно, в многопоточной реализации программа выполняется быстрее пропорционально количеству потоков (в нашем случае в 11 раз быстрее).

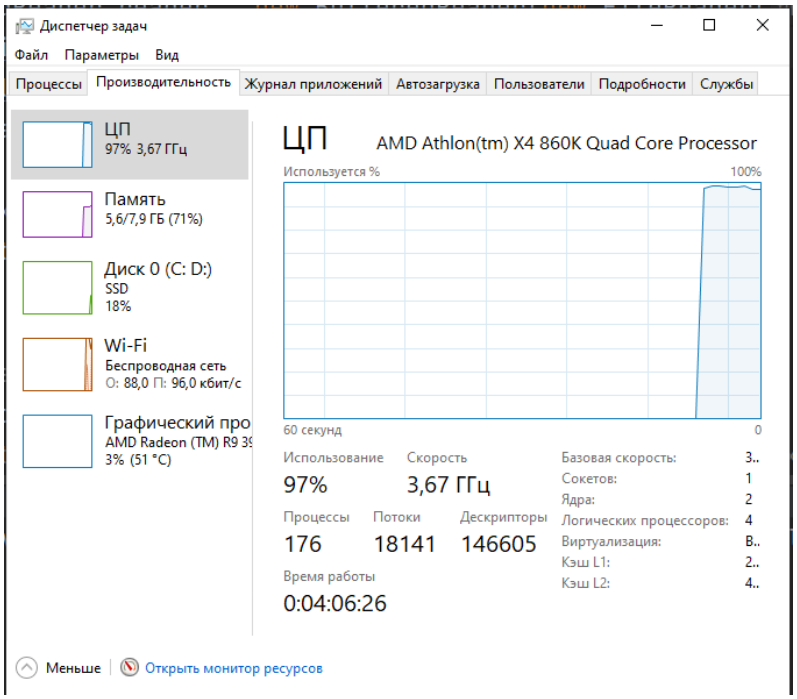
6. Количество потоков, при котором программа перестает работать(проверка на разных Процессорах):

AMD Ryzen 3600:



Максимальное зафиксированное количество потоков (83047)

AMD Athlon X4 860K:



Максимальное зафиксированное количество потоков (18141)

Apple silicon M1 Pro:

Максимальное зафиксированное количество потоков (63789)

7. Анализ результатов работы приложения.

Как и ожидалось реализация через потоки оказалась эффективной для данной задачи. При реализации не возникло каких-либо ошибок, программа работает в соответствии с ожиданием.
Задача выполнена успешно. Все выводы соответствуют нашим ожиданиям. При увеличении числа машин, программа работает стабильно и без ошибок.

8. Выводы по проведенной работе.

В результате разработки многопоточного приложения для заправочной станции достигнута эффективная параллельная обработка запросов. Программа моделирует одновременную заправку нескольких машин и оплату топлива, используя отдельные потоки для каждой бензоколонки. Реализованная синхронизация позволяет эффективно обслуживать клиентов и моделировать различные сценарии использования. Преимущества многопоточности проявляются в сравнении с однопоточной реализацией, демонстрируя улучшенную производительность и возможность параллельной обработки запросов. В дальнейшем приложение предоставляет возможность для исследования работы на различных конфигурациях и настройках приоритетов.

9. Приложение. Листинг программной реализации с комментариями.

Код программы GasStation

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

// Класс представляющий автомобиль
class Car {
    String fuelType;
    double money;
    double litersNeeded;
    int id;
    static int nextId = 1;
```

```
    Car(String fuelType, double money, double litersNeeded) {
        this.fuelType = fuelType;
        this.money = money;
        this.litersNeeded = litersNeeded;
        this.id = nextId++;
    }
}

// Класс представляющий очередь машин для заправки
class CarQueue {
    Queue<Car> carQueue;
    String fuelType;
    List<GasPump> pumps;

    CarQueue(String fuelType) {
        this.fuelType = fuelType;
    }

    // Метод для проверки, свободны ли все колонки для заправки
    boolean isQueuesFree() {
        for (GasPump pump : pumps) {
            if (!pump.isFree) {
                return false;
            }
        }
        return true;
    }

    // Метод для добавления машины в очередь
    void addCarToQueue(Car car) {
        int minimumLength = Integer.MAX_VALUE;
        GasPump best_pump = null;
        for (GasPump pump : pumps) {
            int isNotFree = pump.isFree ? 0 : 1;
            if ((pump.carQueue.size() + isNotFree) < minimumLength) {
                minimumLength = pump.carQueue.size() + isNotFree;
                best_pump = pump;
            }
        }
        if (best_pump == null)
            System.out.println("Ошибка при добавлении автомобиля в очередь");
        else
            best_pump.carQueue.add(car);
    }
}

// Класс представляющий бензоколонку
class GasPump {
    String fuelType;
    Queue<Car> carQueue;
```

```
    double pricePerLiter;
    double litersAvailable;
    boolean isFree;
    Thread pumpThread;

    GasPump(String fuelType, double pricePerLiter, double litersAvailable) {
        this.fuelType = fuelType;
        this.pricePerLiter = pricePerLiter;
        this.litersAvailable = litersAvailable;
        this.isFree = true;
        this.carQueue = new LinkedList<>();
    }

    // Метод для уменьшения счетчика i при завершении заправки
    public synchronized void decrementI() {
        GasStation.lock.lock();
        try {
            GasStation.i--;
            if (GasStation.i == 0) {
                System.out.printf("Время выполнения %d сек.%n",
                        (TimeUnit.SECONDS.convert(System.nanoTime() -
GasStation.startTime, TimeUnit.NANOSECONDS)));
            }
        } finally {
            GasStation.lock.unlock();
        }
    }

    // Метод для проверки возможности заправки машины
    boolean check(Car car) {
        double cost = car.litersNeeded * pricePerLiter;
        if (this.litersAvailable < car.litersNeeded) {
            System.out.printf("На станции недостаточно топлива %s для Car %s.
Имеется %s литров.%n",
                    car.fuelType, car.id, this.litersAvailable);
            return false;
        } else if (car.money >= cost) {
            double change = car.money - cost;
            if (!car.fuelType.equals("накачка"))
                System.out.printf("Автомобиль %d оплатил заправку, сдача = %.2f
руб%n", car.id, change);
            System.out.printf("Автомобиль %d начал заправку %n", car.id,
Thread.currentThread().getId());
            return true;
        } else {
            System.out.printf("У автомобиля %d не хватает денег на заправку.%n",
car.id);
            return false;
        }
    }
```

```
    // Метод для начала заправки
    void startRefueling() {
        while (true) {
            try {
                TimeUnit.MILLISECONDS.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            synchronized (this) {
                if (isFree && !carQueue.isEmpty()) {
                    Car car = carQueue.poll();
                    if (check(car)) {
                        isFree = false;
                        try {
                            this.litersAvailable -= car.litersNeeded;
                            TimeUnit.MILLISECONDS.sleep((int) (car.litersNeeded * 500));
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                        if (car.fuelType.equals("накачка")) {
                            System.out.printf("Колеса автомобиля %d успешно накачаны
%n", car.id);
                        } else {
                            System.out.printf("Автомобиль %d успешно заправлен%n",
car.id);
                            decrementI();
                        }
                        isFree = true;
                    }
                }
            }
        }
    }
}

// Класс представляющий бензоколонку
class GasStation {
    static long startTime;
    static long itsTime;
    static volatile int i;
    static Lock lock = new ReentrantLock();
    private final Map<String, List<GasPump>> pumps;
    private final Map<String, CarQueue> carQueues;

    GasStation() {
        this.pumps = new HashMap<>();
        this.carQueues = new HashMap<>();
        for (List<GasPump> pump_list : pumps.values()) { // параллельная работа
бензоколонки
            for (GasPump pump : pump_list) {
                pump.pumpThread = new Thread(pump::startRefueling);
```

```
                pump.pumpThread.start();
            }
        }
    }

    // Метод для увеличения счетчика i при прибытии машины на заправку
    public synchronized void incrementI() {
        GasStation.lock.lock();
        try {
            GasStation.i++;
        } finally {
            GasStation.lock.unlock();
        }
    }

    // Метод для добавления бензоколонки и очереди на станцию
    void add_pump(String fuelType, double pricePerLiter, double litersAvailable) {
        GasPump pump = new GasPump(fuelType, pricePerLiter, litersAvailable);
        if (pumps.containsKey(fuelType))
            pumps.get(fuelType).add(pump);
        else
            pumps.put(fuelType, new ArrayList<>(List.of(pump)));

        if (!carQueues.containsKey(fuelType)) {
            carQueues.put(fuelType, new CarQueue(fuelType));
            carQueues.get(fuelType).pumps = new ArrayList<>(List.of(pump));
        } else
            carQueues.get(fuelType).pumps.add(pump);

        pump.pumpThread = new Thread(pump::startRefueling);
        pump.pumpThread.start();
    }

    // Метод для прибытия машины на станцию
    void arrive(Car car) {
        if (pumps.containsKey(car.fuelType)) {
            if (car.fuelType.equals("накачка")) {
                System.out.printf("Автомобиль %d прибывает на станцию за
воздухом.%n", car.id);
            } else {
                System.out.printf("Автомобиль %d прибывает на станцию за %s
топливом.%n", car.id, car.fuelType);
                itsTime = (int) System.nanoTime();
                if (i == 0) {
                    startTime = (long) System.nanoTime();
                    System.out.println("Setting start time: " + startTime + "  " + (long)
System.nanoTime());
                }
                incrementI();
                carQueues.get(car.fuelType).addCarToQueue(car);
            }
```

```
        } else {
            System.out.printf("Тип топлива %s недоступен на этой станции.%n",
car.fuelType);
        }
    }
}

// Основной класс программы
public class Main {
    public static void main(String[] args) {
        GasStation station = new GasStation();
        multiThread(station);  // Запуск многопоточной симуляции работы
бензоколонки

        // Вывод информации о бензоколонке
        Map<String, List<GasPump>> pumps = station.get_pumps();
        System.out.println("Заправочная станция");
        for (List<GasPump> pump_list : pumps.values()) {
            for (GasPump pump : pump_list) {
                if (pump.fuelType.equals("накачка"))
                    System.out.println("Накачка шин бесплатно в любых объемах");
                else
                    System.out.println("Тип топлива: " + pump.fuelType + ";  цена за
Литр: " + pump.pricePerLiter
                            + " руб" + ";  Имеется литров на станции: " +
pump.litersAvailable);
            }
        }

        // Обработка ввода из файла
        try (BufferedReader reader = new BufferedReader(new
FileReader("C:\\Users\\Dima\\Desktop\\os\\car.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                processInputLine(line, station);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Принятие ввода с консоли
        Scanner consoleScanner = new Scanner(System.in);
        System.out.println("Введите данные (для завершения введите 'exit'):");

        while (true) {
            String inputLine = consoleScanner.nextLine();
            if (inputLine.equals("exit")) {
                break;
            }
            processInputLine(inputLine, station);
        }
```

```
    }

    // Примеры различных сценариев работы бензоколонки
    private static void oneThread(GasStation station) {
        station.add_pump("92", 10, 100000000);
        for (int i = 0; i <= 10; i++) {
            station.arrive(new Car("92", 100000, 10));
        }
    }

    private static void multiThread(GasStation station) {
        for (int i = 0; i <= 10; i++) {
            station.add_pump("92", 10, 100000000);
        }
        for (int i = 0; i <= 10; i++) {
            station.arrive(new Car("92", 100000, 10));
        }
    }

    private static void fall(GasStation station) {
        for (int i = 0; i <= 100000; i++) {
            station.add_pump("92", 10, 100000000);
        }
        for (int i = 0; i <= 100000; i++) {
            station.arrive(new Car("92", 100000, 100));
        }
    }

    // Метод для обработки входной строки
    private static void processInputLine(String inputLine, GasStation station) {
        Scanner scanner = new Scanner(System.in);
        Map<String, List<GasPump>> pumps = station.get_pumps();
        String[] parts = inputLine.split("\\s+");
        if (parts.length == 3 || (parts.length == 2 && parts[0].equals("накачка"))) {
            // Обработка ввода для машины
            // ...
        } else if (parts.length == 1 && Objects.equals(parts[0], "fuel")) {
            // Вывод информации о бензоколонке
            // ...
        } else if (parts.length == 4 && Objects.equals(parts[0], "add_pump")) {
            // Добавление бензоколонки
            // ...
        } else {
            System.out.println("Неверные входные данные! Правильный вариант: -
\"<тип топлива> <деньги> <необходимые литры>\"");
        }
    }
}

// Класс для вывода информации о потоках
class ThreadInfo {
```

```
    public static void printThreadInfo() {
        int processors = Runtime.getRuntime().availableProcessors();
        System.out.println("Количество доступных процессоров: " + processors);

        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        int activeThreads = Thread.activeCount();
        System.out.println("Количество активных потоков: " + activeThreads);
    }
}

// Класс для вывода информации о процессорах
class ProcessorInfo {
    public static void printProcessorInfo() {
        int processors = Runtime.getRuntime().availableProcessors();
        System.out.println("Количество доступных процессоров: " + processors);
    }
}
```