Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РУБЕЖНЫЙ КОНТРОЛЬ №1
по курсу «Анализ алгоритмов»

«Моделирование конвейерной обработки»

Студент Маслова Марина Дмитриевна

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватель Волкова Лилия Леонидовна

2021 г.

Содержание

Задание

На вход программе подются следующие значения:

– N – натуральное число, количество обрабатываемых заявок;

– $t^i_{\text{М.О.}}$, где $i \in \{1, 2, 3\}$ – среднее время обработки заявки на i-ой ленте;

– Δt_i, где $i \in \{1, 2, 3\}$ – разброс значений времени обработки заявки на i-ой ленте.

1 Аналитическая часть

В данном разделе представлено теоретическое описание моделирования конвейерной обработки данных.

1.1 Конвейерная обработка данных

Модель конвейерной обработки данных была описана в лабораторной работе №5 [1], в данной работе, так как на вход программе подаются значения времени обработки заявок на каждой из лент, на этапах не будет выполняться никакой полезной работы, на каждом потоке будет смоделирована обработка с помощью задержки на заранее сгенерированное время обработки заявки, попадающее в заданный диапазон.

1.2 Определение времени простоя лент

Время работы каждой ленты конвейера затрачивается на одну из следующих операций:

– обработка заявки;

– обслуживающие действия (получение заявки из входной очереди, отправка заявки на следующий этап);

– простой (ожидание следующей заявки).

Будем считать, что лента начинает работать с момента приема первой заявки $t^i_{\text{нач}}$ и до момента окончания обработки последней заявки $t^i_{\text{кон}}$.

Таким образом, время простоя каждой ленты можно вычислить по формуле 1.1:

$$t_{\text{простоя}} = t^i_{\text{кон}} - t^i_{\text{нач}} - t_{\text{обработки}} - t_{\text{обслуживания}} \tag{1.1}$$

1.3 Вывод

В данном разделе была кратко описана модель реализуемой конвейерной обработки, также было представлено математическое описание поиска времени простоя каждой из лент. Из представленных описаний можно предъявить следующие требования к разрабатываемому программному обеспечению: по заданным количеству заявок и времени обработки заявки на каждой ленте

программа должна выдавать лог работы каждой ленты, а также время их простоя, при неверных входных данных программа должна выдавать сообщение об ошибке.

2 Конструкторская часть

В данном разделе разрабатываются алгоритмы каждого потока конвейерной обрабокти, включая главный, также описывается структура программы и способы её тестироваия.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема главного потока, запускающего потоки этапов конвейера, схемы алгоритмов которых представлены на рисунках 2.2-2.4.

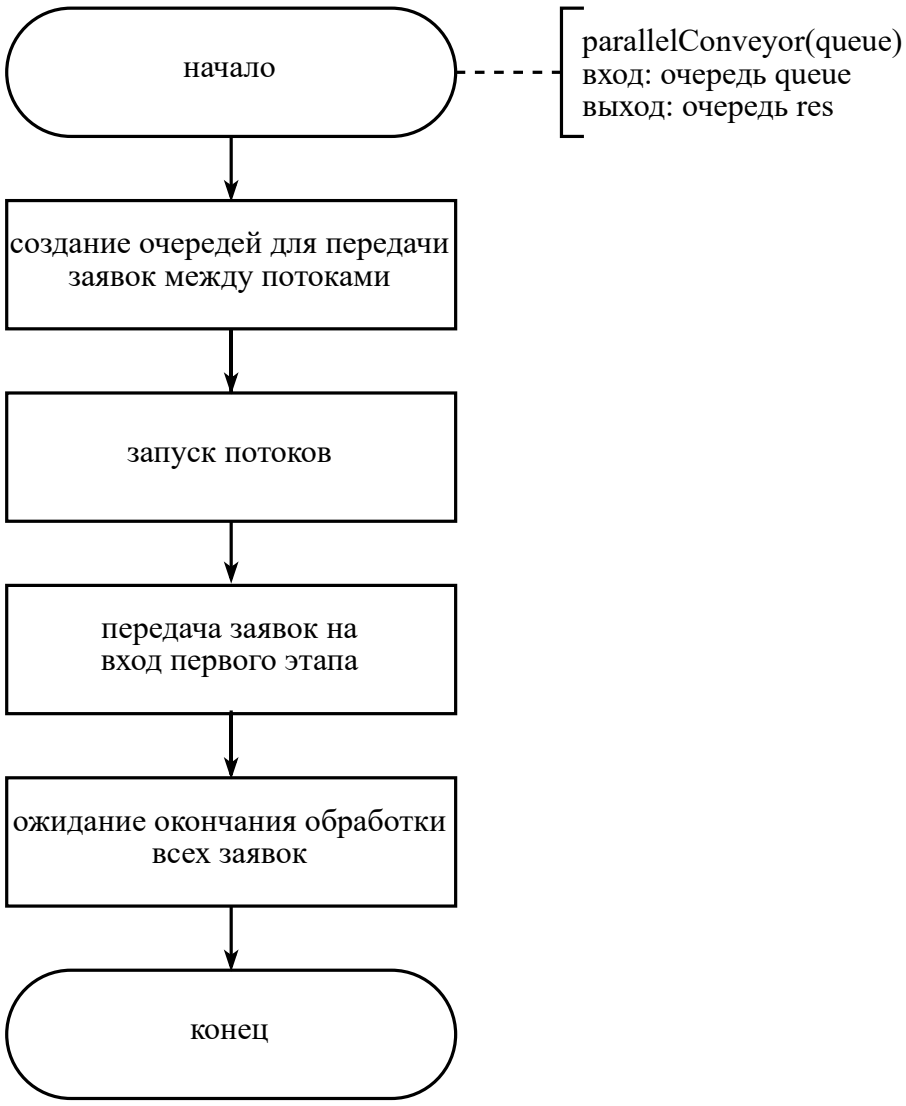

Рисунок 2.1 – Схема главного потока конвейерной обработки заявок

Рисунок 2.2 – Схема потока первого этапа обработки

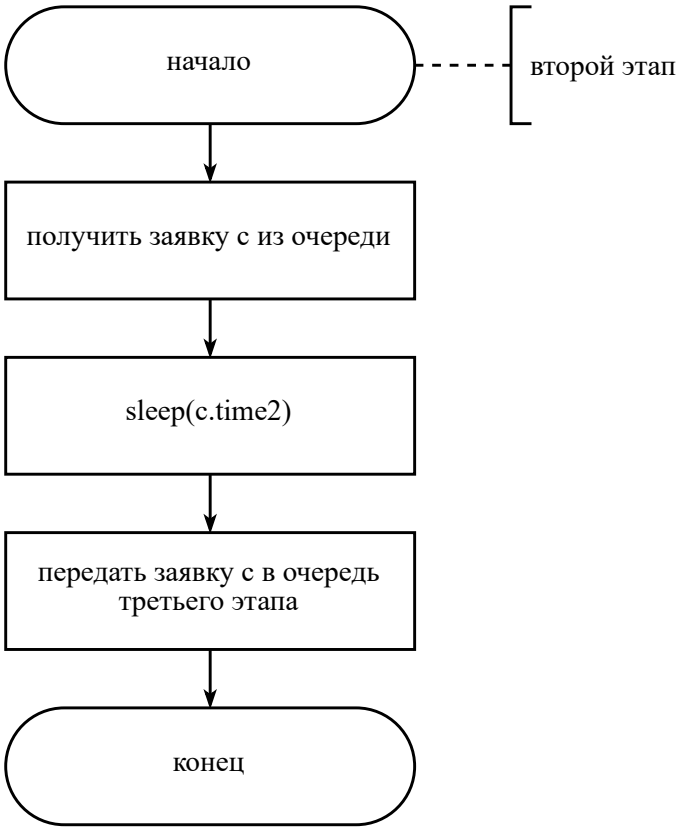

Рисунок 2.3 – Схема потока второго этапа обработки

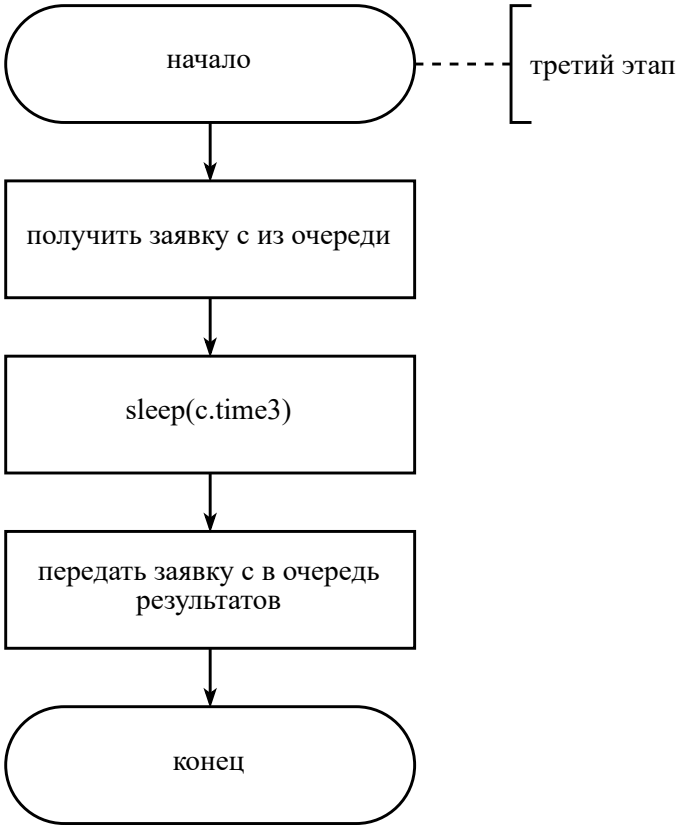

Рисунок 2.4 – Схема потока третьего этапа обработки

2.2 Структура разрабатываемого ПО

Для реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Каждый из алгоритмов будет представлен отдельной функцией, при необходимости будут выделены подпрограммы для каждой из них. Также будут реализованы функции для ввода-вывода и функция, вызывающая все подпрограммы для связности и полноценности программы.

2.3 Классы эквивалентности при тестировании

Для тестирования программного обеспечения во множестве тестов будут выделены следующие классы эквивалентности:

- отрицательное число;
- ноль заявок;
- нечисловые данные;
- отрицательное время;
- нулевое время;

– нулевой разброс;
– простой тест.

2.4 Вывод

В данном разделе были разработаны алгоритмы конвейерной обработки заявок, была описана структура разрабатываемого ПО. Для дальнейшей проверки правильности работы программы были выделены классы эквивалентности тестов.

3 Технологическая часть

В данном разделе описаны требования к программному обеспечению, средства реализации, приведены листинги кода и данные, на которых будет проводиться тестирование.

3.1 Требования к программному обеспечению

Программа должна предоставлять следующие возможности:

- ввода количества обрабатываемых заявок, среднего времени и его разброса для каждой ленты (в миллисекундах);
- вывода лога работы программы и время простоя каждой ленты;
- получения времени обработки всех заявок.

3.2 Средства реализации

Для реализации данной лабораторной работы выбран компилируемый многопоточный язык програмирования `Go`[2], так как он предоставляет необходимый функционал для работы с потоками и простой реализациии их взаимодействия. Интерпретируемый язык программирования высокого уровня Python[3] был выбран для визуализации данных эксперимента, так как он предоставляет большое число настроек параметров графика с использованием простого синтаксиса.

В качестве среды разработки выбран текстовый редактор Vim[4] с установленными плагинами автодополнения и поиска ошибок в процессе написания, так как он реализует быстрое перемещение по тексту программы и простое взаимодействие с командной строкой.

Замеры времени проводились при помощи функции `Now()` из библиотеки `time`[5].

3.3 Листинги кода

В данном подразделе представлены листинги кода алгоритмов:

– генерация времени обработки заявки на каждой ленте (листинг 3.1);

– параллельная реализация конвейера (листинг 3.2);

– получение лога работы программы (листинг 3.3).

Листинг 3.1 – Генерация времени обрабоки заявки на каждой ленте

```
func genIntAB(a int, b int) int {
    return a + rand.Intn(b-a+1)
}

func GenerateRequests(num int, avgTimes [3]int, avgDeltaTimes [3]int) *Queue {
    queue := CreateQueue(num)

    for i := 0; i < num; i++ {
        sleepy := new(Sleepyhead)

        sleepy.SetTime1(time.Duration(genIntAB(avgTimes[0]-avgDeltaTimes[0],
            avgTimes[0]+avgDeltaTimes[0])) * time.Millisecond)
        sleepy.SetTime2(time.Duration(genIntAB(avgTimes[1]-avgDeltaTimes[1],
            avgTimes[1]+avgDeltaTimes[1])) * time.Millisecond)
        sleepy.SetTime3(time.Duration(genIntAB(avgTimes[2]-avgDeltaTimes[2],
            avgTimes[2]+avgDeltaTimes[2])) * time.Millisecond)

        queue.Push(sleepy)
    }

    return queue
}
```

Листинг 3.2 – Параллельная реализация конвейера

```
func ParallelConveyor(inQueue *Queue) (*Queue, time.Duration) {
    mainToF := make(chan *Sleepyhead, inQueue.capacity)
    fToS := make(chan *Sleepyhead, inQueue.capacity)
    sToT := make(chan *Sleepyhead, inQueue.capacity)
    tToMain := make(chan int)
    res := CreateQueue(inQueue.capacity)

    first := func() {
        for {
            select {
            case c := <-mainToF:
                c.start1 = time.Now()
```

Листинг 3.2 (продолжение)

```
                time.Sleep(c.time1)
                c.end1 = time.Now()

                fToS <- c
            }
        }
    }

    second := func() {
        for {
            select {
            case c := <-fToS:
                c.start2 = time.Now()
                time.Sleep(c.time2)
                c.end2 = time.Now()

                sToT <- c

            }
        }
    }

    third := func() {
        for {
            select {
            case c := <-sToT:
                c.start3 = time.Now()
                time.Sleep(c.time3)
                c.end3 = time.Now()

                res.Push(c)
                if res.tail == res.capacity-1 {
                    tToMain <- 0
                }
            }
        }
    }

    go first()
    go second()
    go third()

    cip := inQueue.Pop()
    for cip != nil {
        mainToF <- cip
        cip = inQueue.Pop()
    }
```

Листинг 3.2 (продолжение)

```
60
61      start := time.Now()
62      <-tToMain
63      fullTime := time.Now().Sub(start)
64
65      return res, fullTime
66  }
```

Листинг 3.3 – Получение лога работы программы

```
1  func PrintLog(full time.Duration, q *Queue, needSort bool,
2      needTable bool) (time.Duration, time.Duration, time.Duration) {
3      reqTime1 := time.Duration(0)
4      reqTime2 := time.Duration(0)
5      reqTime3 := time.Duration(0)
6
7      tmp := new(Queue)
8      tmp.GetCopyBy(q)
9      log := make([]logRow, q.capacity*3, q.capacity*3)
10
11     i := 0
12     j := 0
13     c := tmp.Pop()
14     begin := c.start1
15     for c != nil {
16         log[j] = logRow{
17             requestNum: i,
18             stage:      1,
19             start:      c.start1.Sub(begin),
20             end:        c.end1.Sub(begin)}
21         reqTime1 += log[j].end - log[j].start
22         j++
23
24         log[j] = logRow{
25             requestNum: i,
26             stage:      2,
27             start:      c.start2.Sub(begin),
28             end:        c.end2.Sub(begin)}
29         reqTime2 += log[j].end - log[j].start
30         j++
31
32         log[j] = logRow{
33             requestNum: i,
34             stage:      3,
35             start:      c.start3.Sub(begin),
```

Листинг 3.3 (продолжение)

```
            end:        c.end3.Sub(begin)}
        reqTime3 += log[j].end - log[j].start
        i++
        j++

        c = tmp.Pop()
    }

    if needSort {
        sort.Slice(log, func(i, j int) bool {
            f := log[i].stage == log[j].stage

            if f {
                return log[i].start < log[j].start
            }

            return log[i].stage < log[j].stage
        })
    } else {
        sort.Slice(log, func(i, j int) bool {
            return log[i].start < log[j].start
        })
    }

    if needTable {
        table := tablewriter.NewWriter(os.Stdout)
        table.SetHeader([]string{"#", "Stage", "Begin", "End"})

        for _, v := range log {
            f := fmt.Sprintf("%v", v.requestNum)
            s := fmt.Sprintf("%v", v.stage)
            t := fmt.Sprintf("%v", v.start)
            fo := fmt.Sprintf("%v", v.end)
            table.Append([]string{f, s, t, fo})
        }

        table.Render()
        fmt.Println("TOTAL: ", full)
    }

    fullLine1Time := log[q.capacity-1].end - log[0].start
    fullLine2Time := log[q.capacity*2-1].end - log[q.capacity].start
    fullLine3Time := log[q.capacity*3-1].end - log[q.capacity*2].start

    downTime1 := fullLine1Time - reqTime1
    downTime2 := fullLine2Time - reqTime2
    downTime3 := fullLine3Time - reqTime3
```

Листинг 3.3 (продолжение)

```
83
84      fmt.Println("Время простоя 1 ленты:", downTime1)
85      fmt.Println("Время простоя 2 ленты:", downTime2)
86      fmt.Println("Время простоя 3 ленты:", downTime3)
87
88      return downTime1, downTime2, downTime3
89  }
```

3.4 Описание тестирования

В таблице 3.1 приведены функциональные тесты программы.

Таблица 3.1 – Функциональные тесты

Количество заявок	**Ожидаемый результат**
-1	Сообщение об ошибке
0	Сообщение об ошибке
j	Сообщение об ошибке
10 -1	Сообщение об ошибке
10 0	Сообщение об ошибке
10 100 0	Лог работы программы
10 100 10	Лог работы программы

3.5 Вывод

В данном разделе были реализовано моделирование конвейрной обработки данных. Также были написаны тесты для каждого класса эквивалентности, описанного в конструкторском разделе.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

– Операционная система: Manjaro [6] Linux x86_64.

– Память: 8 GiB.

– Процессор: Intel® Core™ i5-8265U, 4 физических ядра, 8 логических ядра[7].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, окружением, а также непосредственно системой тестирования.

4.2 Примеры работы программы

На рисунке 4.1 представлены результаты работы программы.

4.3 Результаты тестирования

Программа была протестирована на входных данных, приведенных в таблице 3.1. Полученные результаты работы программы совпали с ожидаемыми результатами.

4.4 Постановка эксперимента по замеру времени

Для оценки зависимости времени простоя каждой ленты от разброса заданных значений времени и соотношениями между средними значениями времени обработки заявок был проведен эксперимент. Так как каналы для передачи заявок между потоками не позволяют получить время обслуживающих действий, для получения достоверных результатов необходимо такое время обработки заявок, для которого время обслуживающие действия мало.

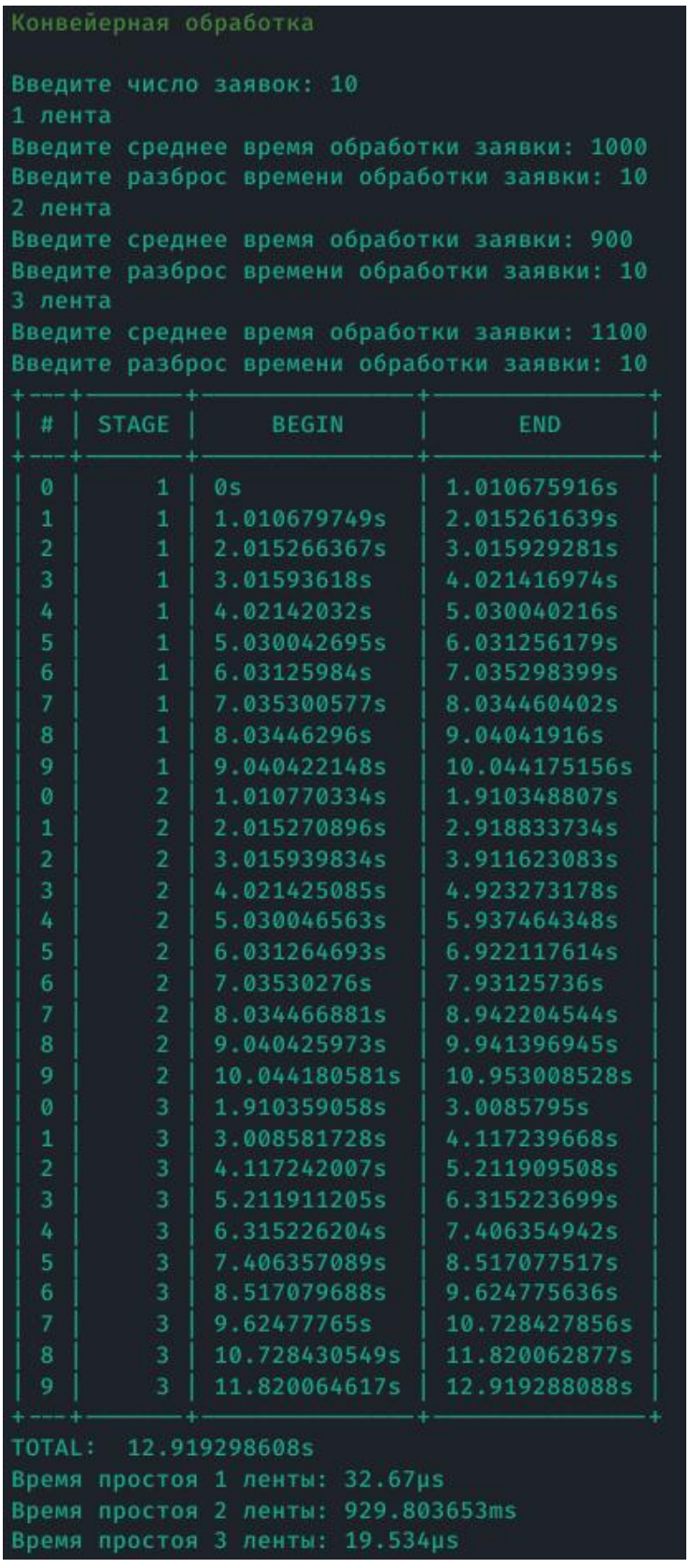

Рисунок 4.1 – Пример работы программы

Для этого программе подаются различные порядки среднего времени обработки заявки (одинаковые для всех трех лент без разброса для исключения ситуации простоя), находится разница между ожидаемым временем обработки

всех заявок и полученным, если эта разница мала по сравнению со значением ожидаемого времени, то заданный порядок можно использовать для анализа (эксперимент 1).

Эксперимент по оценке зависимости времени простоя лент от разброса был проведен на одинаковых значениях среднего времени обрабоки на всех лентах и разбросах от 10 до 810 с шагом 100 (в миллисекундах) (эксперимент 2).

Для анализа зависимости времени простоя от соотношений средних значений времени обработки программа была протестирована на следующих соотноешниях: 1:1:1, 2:1:1, 1:2:1, 1:1:2, 2:2:1, 1:2:2, 2:1:2 (эксперимент 3).

Результаты эксперимента были представлены в виде таблиц и графиков, приведенных в следующем подразделе.

4.5 Результаты эксперимента

На листинге 4.1 представлены результаты провдения эксперимента 1.

На листинге 4.2 представлены результаты провдения эксперимента 2. На рисунке 4.2 представлены графики зависимости простоя 2-ой и 3-ей лент (первая лента не представлена потому, что она не простаивает, так как все заявки сразу передаются в её входную очередь).

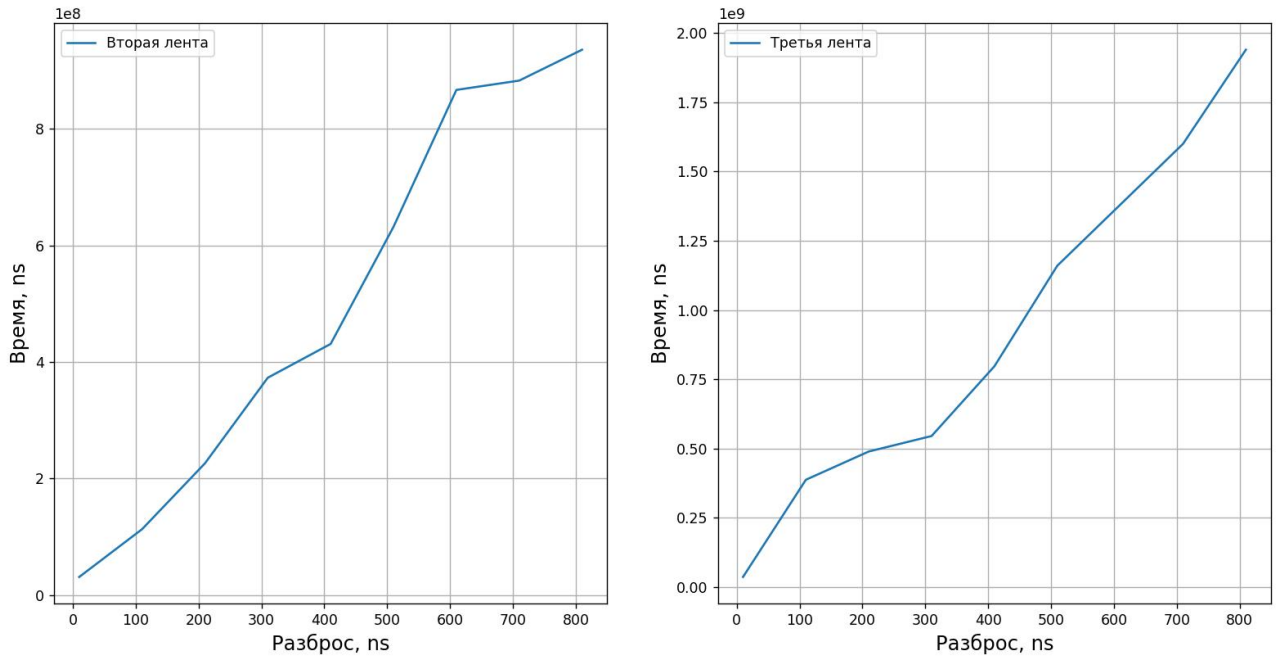

Рисунок 4.2 – Графики зависимости времени простоя второй и третьей лент

На листинге 4.3 представлены результаты провдения эксперимента 3.

Листинг 4.1 – Результаты эксперимента по определению порядка среднего времени

```
=== RUN   TestFindPrepare
    conveyor_test.go:18: True = 12ms
    conveyor_test.go:19: Got = 13.781929ms
    conveyor_test.go:20: Dif = 1.781929ms
    conveyor_test.go:18: True = 120ms
    conveyor_test.go:19: Got = 123.489534ms
    conveyor_test.go:20: Dif = 3.489534ms
    conveyor_test.go:18: True = 1.2s
    conveyor_test.go:19: Got = 1.208237172s
    conveyor_test.go:20: Dif = 8.237172ms
    conveyor_test.go:18: True = 12s
    conveyor_test.go:19: Got = 12.006658648s
    conveyor_test.go:20: Dif = 6.658648ms
--- PASS: TestFindPrepare (13.35s)
```

Листинг 4.2 – Результаты эксперимента по определению зависимости времени простоя от разброса средних значений времени

```
=== RUN   TestFindPrepare
    conveyor_test.go:18: True = 12ms
    conveyor_test.go:19: Got = 13.781929ms
    conveyor_test.go:20: Dif = 1.781929ms
    conveyor_test.go:18: True = 120ms
    conveyor_test.go:19: Got = 123.489534ms
    conveyor_test.go:20: Dif = 3.489534ms
    conveyor_test.go:18: True = 1.2s
    conveyor_test.go:19: Got = 1.208237172s
    conveyor_test.go:20: Dif = 8.237172ms
    conveyor_test.go:18: True = 12s
    conveyor_test.go:19: Got = 12.006658648s
    conveyor_test.go:20: Dif = 6.658648ms
--- PASS: TestFindPrepare (13.35s)
=== RUN   TestDeltas
    conveyor_test.go:30: delta =  10
Время простоя 1 ленты: 26.633us
Время простоя 2 ленты: 28.412178ms
Время простоя 3 ленты: 31.317518ms
    conveyor_test.go:30: delta =  110
Время простоя 1 ленты: 24.814us
Время простоя 2 ленты: 90.70415ms
Время простоя 3 ленты: 560.451845ms
    conveyor_test.go:30: delta =  210
```

Листинг 4.2 (продолжение)

```
Время простоя 1 ленты: 25.604us
Время простоя 2 ленты: 130.179379ms
Время простоя 3 ленты: 77.31079ms
    conveyor_test.go:30: delta =  310
Время простоя 1 ленты: 24.928us
Время простоя 2 ленты: 60.711549ms
Время простоя 3 ленты: 464.033863ms
    conveyor_test.go:30: delta =  410
Время простоя 1 ленты: 27.039us
Время простоя 2 ленты: 1.970707632s
Время простоя 3 ленты: 1.468260007s
    conveyor_test.go:30: delta =  510
```

Листинг 4.3 – Результаты эксперимента по определению зависимости времени простоя каждой ленты от соотношения средних значений времени

```
=== RUN   Test111
Время простоя 1 ленты: 21.8us
Время простоя 2 ленты: 107.493295ms
Время простоя 3 ленты: 353.217231ms
--- PASS: Test111 (12.63s)
=== RUN   Test211
Время простоя 1 ленты: 34.056us
Время простоя 2 ленты: 8.718466767s
Время простоя 3 ленты: 8.681591645s
--- PASS: Test211 (21.84s)
=== RUN   Test121
Время простоя 1 ленты: 24.018us
Время простоя 2 ленты: 34.389us
Время простоя 3 ленты: 9.13371603s
--- PASS: Test121 (21.61s)
=== RUN   Test112
Время простоя 1 ленты: 28.391us
Время простоя 2 ленты: 824.26088ms
Время простоя 3 ленты: 19.746us
--- PASS: Test112 (21.82s)
=== RUN   Test221
Время простоя 1 ленты: 25.106us
Время простоя 2 ленты: 164.800087ms
Время простоя 3 ленты: 9.687310433s
--- PASS: Test221 (22.98s)
=== RUN   Test122
Время простоя 1 ленты: 28.599us
```

Листинг 4.3 (продолжение)

```
Время простоя 2 ленты: 27.06us
Время простоя 3 ленты: 22.949us
--- PASS: Test122 (23.31s)
=== RUN   Test212
Время простоя 1 ленты: 34.775us
Время простоя 2 ленты: 8.684922604s
Время простоя 3 ленты: 434.260608ms
--- PASS: Test212 (23.56s)
```

4.6 Вывод

По результатам эксперимента можно сделать следующие выводы:

- на данной машине время обслуживающие действия становятся мало при времени обработки заявок от 1 секуды;
- при увеличении разброса значений времени обработки заявок увеличивается время простоя второй и третьей лент;
- первая лента не простаивает, так как все заявки подаются на её входную очередь одновременно;
- при соотношении средних значений времени обработки заявки 1:1:1 вторая и третья лента простаивают малое время за счет обрабатывающих действий на первой ленте;
- при соотношении средних значений времени обработки заявки 2:1:1 вторая и третья лента простаивают значительное время, так как после обработки поданной заяки ожидают, пока свою обработку завершит первая лента;
- при соотношении средних значений времени обработки заявки 1:2:1 простаивает третья лента, так как ждет пока заявку обработает вторая лента;
- при соотношении средних значений времени обработки заявки 1:1:2 ленты не простаивают, так как третья лента передает свои заявки на выход конвейера и завершения ею обрабоки заявки другие ленты не ждут;
- при соотношении средних значений времени обработки заявки 2:2:1 простаивает третья лента, так как ждет завершения обработки заявок первыми двумя лентами;

– при соотношении средних значений времени обработки заявки 1:2:2 ленты не простаивают;
– при соотношении средних значений времени обработки заявки 2:1:2 простаивает 2 лента, так ждет завершения обработки заявок первой лентой.

Заключение

Таким образом, для организации эффективной работы конвейера время обработки заявки на каждой ленте должны быть приблизительно равны. В ином случае для исключения простоя лент необходимо подключать к обработке дополнительные потоки для тех лент, время обработки заявок на которых больше.

Список литературы

[1] Маслова М. Д. Отчет по лабораторной работе №5 по курсу "Анализ алгоритмов". Режим доступа: https://github.com/MyMiDiII/bmstu-aa/blob/main/lab05/docs/pdf/report.pdf (дата обращения: 11.12.2021).

[2] The Go Programming Language [Электронный ресурс]. Режим доступа: https://golang.org/ (дата обращения: 07.12.2021).

[3] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 12.10.2021).

[4] welcome home : vim online [Электронный ресурс]. Режим доступа: https://www.vim.org/ (дата обращения: 12.10.2021).

[5] Package time [Электронный ресурс]. Режим доступа: https://golang.org/pkg/time/ (дата обращения: 07.12.2021).

[6] Manjaro — enjoy the simplicity [Электронный ресурс]. Режим доступа: https://manjaro.org/ (дата обращения: 17.10.2021).

[7] Процессор Intel® Core™ i5-8265U [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/149088/intel-core-i5-8265u-processor-6m-cache-up-to-3-90-ghz.html (дата обращения: 17.10.2021).