

*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)*

ОТЧЕТ

По лабораторной работе №5
По курсу: «Анализ алгоритмов»
Тема: «Конвейер»

Студент:	Козаченко А. А.
Группа:	ИУ7-54Б
Преподаватели:	Волкова Л. Л., Строганов Ю. В.

Москва, 2019

Оглавление

Введение	2
1 Аналитическая часть	3
1.1 Описание конвейерной обработки данных	3
2 Конструкторская часть	4
2.1 Разработка конвейерной обработки данных	4
3 Технологическая часть	5
3.1 Средства реализации	5
3.2 Листинг кода	5
3.3 Тестирование функций	7
4 Исследовательская часть	9
4.1 Время выполнения	9
Заключение	10
Литература	10

Введение

При обработке данных могут возникать ситуации, когда необходимо обработать множество данных последовательно несколькими алгоритмами. В этом случае удобно использовать конвейерную обработку данных. Это может быть полезно при следующих задачах:

- шифровании данных;
- сортировки и фильтрации данных;
- и др.

Цель данной работы: получить навык организации асинхронного взаимодействия потоков на примере конвейерной обработки данных.

В рамках выполнения работы необходимо решить следующие задачи:

- рассмотреть и изучить конвейерную обработку данных;
- реализовать конвейер с количеством лент не меньше трех в многопоточной среде;
- на основании проделанной работы сделать выводы.

1 Аналитическая часть

1.1 Описание конвейерной обработки данных

При конвейерной обработке данных каждая лента имеет свою очередь с некоторыми задачами, ожидающими обработки. Лента берет данные из своей очереди с входными данными, проводит с ними необходимые операции и передает в очередь следующей ленты или, в случае последней ленты, в пул обработанных задач.

Вывод

В данной работе стоит задача реализации конвейера.

2 Конструкторская часть

2.1 Разработка конвейерной обработки данных

Принцип работы стадийной обработки представлен на рис. 2.1.

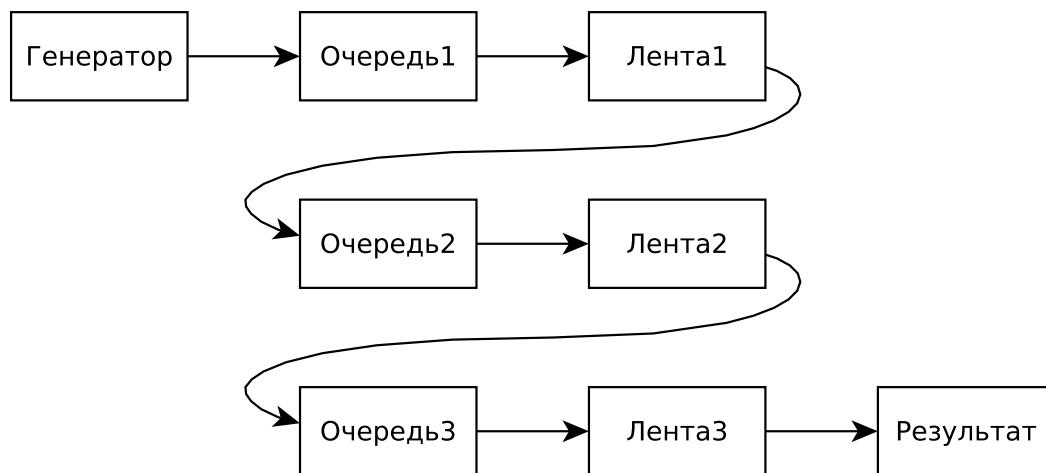


Рис. 2.1: Конвейерная обработка данных

Вывод

Был показан принцип работы конвейерной обработки данных.

3 Технологическая часть

В данном разделе приведены средства реализации и листинг кода.

3.1 Средства реализации

В качестве языка программирования был выбран Go, так как он предоставляет широкие возможности и крайне удобный интерфейс для эффективной реализации асинхронной, параллельной обработки данных [1].

Для измерения времени использовалась стандартная библиотека `time`. Так как основное время работы составляет ожидание `sleep`, то достаточно замерить время работы один раз.

3.2 Листинг кода

В листинге 3.1 представлена реализация конвейерная обработка данных.

```
package main

import (
    "fmt"
    "math/rand"
    "sync"
    "time"
)

const (
    TasksNumber = 3
)

type Task struct {
    Value int
    Result int
    StartTimes []time.Time
    EndTimes []time.Time
}

type job func(in, out chan interface{})

func ExecutePipeline(jobs ...job) {
```

```

wg := &sync.WaitGroup{}
in := make(chan interface{}, TasksNumber)
for i := range jobs {
    out := make(chan interface{}, TasksNumber)
    wg.Add(1)
    go func(function job, in, out chan interface{}) {
        defer wg.Done()
        defer close(out)
        function(in, out)
    }(jobs[i], in, out)
    in = out
}
wg.Wait()
}

func dataGenerator(_, out chan interface{}) {
    s := rand.NewSource(time.Now().UnixNano())
    r := rand.New(s)
    for i := 0; i < TasksNumber; i++ {
        val := r.Intn(100)
        task := Task{Value: val, Result: val}
        out <- task
    }
}

func makeHandler(sleepDuration int, multiplier int) func(chan interface{}, chan
interface{}) {
    return func(in, out chan interface{}) {
        for rawTask := range in {
            task := rawTask.(Task)
            task.StartTimes = append(task.StartTimes, time.Now())

            task.Result *= multiplier
            time.Sleep(time.Millisecond * time.Duration(sleepDuration))

            task.EndTimes = append(task.EndTimes, time.Now())
            out <- task
        }
    }
}

func logHandler(in, _ chan interface{}) {
    for rawTask := range in {
        task := rawTask.(Task)
        fmt.Printf("Task's start value=%d, result=%d\n", task.Value,
            task.Result)
        fmt.Printf("Start 1st: %s, end 1st: %s\n", task.StartTimes[0].Local(),
            task.EndTimes[0].Local())
    }
}

```

```

        fmt.Printf("Start_2nd:_%s, end_2nd:_%s\n", task.StartTimes[1].Local(),
            task.EndTimes[1].Local())
        fmt.Printf("Start_3d:_%s, end_3d:_%s\n", task.StartTimes[2].Local(),
            task.EndTimes[2].Local())
        fmt.Println()
    }
}

func main() {
    jobs := []job{
        job(dataGenerator),
        job(makeHandler(100, 2)),
        job(makeHandler(200, 3)),
        job(makeHandler(300, 2)),
        job(logHandler),
    }

    start := time.Now()
    ExecutePipeline(jobs...)
    end := time.Since(start)

    fmt.Printf("All_time:_%s\n", end)
}

```

Листинг 3.1: Алгоритм сортировки пузырьком

3.3 Тестирование функций

Для тестирования были реализованы функции, представленные в листинге 3.2. Результаты тестирования представлены в таблице 3.1. Видно, что тестирование пройдено успешно.

```

job(func(in, out chan interface{}) {
    out <- uint32(1)
    out <- uint32(3)
    out <- uint32(4)
}),
job(func(in, out chan interface{}) {
    for val := range in {
        out <- val.(uint32) * 3
    }
}),
job(func(in, out chan interface{}) {
    for val := range in {
        fmt.Println("collected", val)
    }
})

```



```
        atomic.AddUint32(&recieved, val.(uint32))
    }
    }),
```

Листинг 3.2: Тестовые задачи

Входные данные	Ожидаемый результат	Результат
1,3,4	24	24

Таблица 3.1: Тестирование конвейерной обработки

Вывод

Правильный выбор инструментов разработки позволил эффективно реализовать алгоритмы, настроить модульное тестирование и выполнить исследовательский раздел лабораторной работы.

4 Исследовательская часть

4.1 Время выполнения

Реализованная контейнерная обработка данных работает за 1,2с. Последовательная реализация потребовала бы $(0,1 + 0,2 + 0,3) * 3 = 1,8$ с, что на 50% медленней.

Вывод

Алгоритм сортировки вставками работает лучше остальных двух на случайных числах и уже отсортированных, практический интерес, конечно, представляет лишь первый случай, на котором сортировка вставками почти на четверть быстрее сортировки пузырьком и на тринадцатую часть быстрее сортировки выбором.

Заключение

В рамках лабораторной работы была рассмотрена и изучена конвейерная обработка данных. Благодаря ней возможна крайне удобная реализация задач, требующих поэтапной обработки некоторого набора данных, а также в некоторых случаях позволяет значительно ускорить выполнение программы (в реализованном синтетическом примере выигрыш составил 50%).

Литература

- [1] Dan Gorby. Multi-thread For Loops Easily and Safely in Go. URL: <https://medium.com/@greenraccoon23/multi-thread-for-loops-easily-and-safely-in-go-a2e915302f8b> (дата обращения: 28.11.2019).