Міністерство освіти і науки України

Національний технічний університет України «КПІ» імені Ігоря Сікорського

Кафедра обчислювальної техніки ФІОТ

ЗВІТ

з розрахунково-графічної роботи

з навчальної дисципліни «Архітектура програмного забезпечення»

Тема:
Архітектурні діаграми та бенчмарки

Виконав
Студенти 3 курсу ІП-94
Рудик М.
Сумський П.
Литвишко Н.

Перевірив
Мазур Р. Ф.

Київ 2021

Мета: Закріплення навичок ілюстрації організації програмних систем та оцінки часу виконання алгоритмів.

Завдання

- Для 2-гої роботи, підтвердьте лінійний час виконання вашої функції перетворення чи обчислення вхідного виразу.
- Для 3-тої роботи, побудуйте діаграму взаємодії компонентів у вашій імплементації.
- Для 4-ої роботи, побудуйте діаграму взаємодії для вашої реалізації (на ній, скоріш за все, мають опинитися компоненти парсера, черги команд, ядра цикла) та підтвердьте лінійний час роботи вашого парсера команд.

Частина перша

Для 2-гої роботи функція для підтвердження лінійного часу виконання нашої функції:

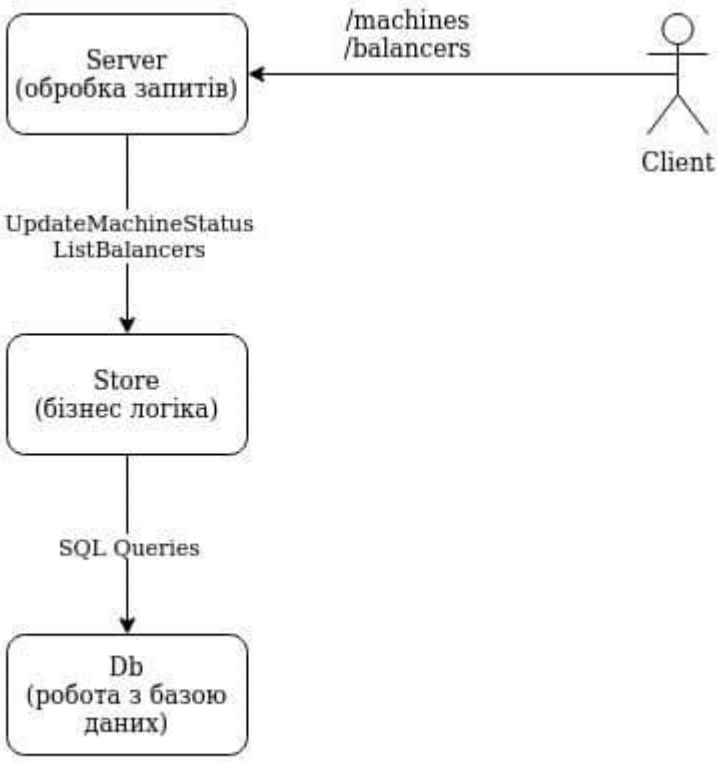
```go
+ func BenchmarkCalculatePostfix(b *testing.B) {
+       postfixes := []string{
+               "4 2 - 3 * 5 +",
+               "9 3 4 + * 6 / 1 - 8 2 ^ 5 1 - + *",
+               "4 2 - 3 * 5 + 4 2 - 3 * 5 + 3 4 2 * 1 3 - 2 ^ / +",
+               "3 4 2 * 1 3 - 2 ^ / + 9 3 4 + * 6 / 1 - 8 2 ^ 5 1 - + *",
+               "1 2 + 4 * 3 + 4 2 - 3 * 6 + 9 3 4 + * 6 / 1 - 8 2 ^ 5 1 - + * 4 2 - 3 * 5 +",
+               "4 2 - 3 * 5 + 9 3 4 + * 6 / 1 - 8 2 ^ 5 1 - + * 4 2 - 3 * 5 + 4 2 - 3 * 5 + 3 4 2
+               "4 2 - 3 * 5 + 4 2 - 3 * 5 + 3 4 2 * 1 3 - 2 ^ / + 3 4 2 * 1 3 - 2 ^ / + 9 3 4 + *
+               "3 4 2 * 1 3 - 2 ^ / + 9 3 4 + * 6 / 1 - 8 2 ^ 5 1 - + * 1 2 + 4 * 3 + 4 2 - 3 * 6
+               "4 2 - 3 * 5 + 9 3 4 + * 6 / 1 - 8 2 ^ 5 1 - + * 4 2 - 3 * 5 + 4 2 - 3 * 5 + 3 4 2
+               "3 4 2 * 1 3 - 2 ^ / + 9 3 4 + * 6 / 1 - 8 2 ^ 5 1 - + * 1 2 + 4 * 3 + 4 2 - 3 * 6
+               "4 2 - 3 * 5 + 9 3 4 + * 6 / 1 - 8 2 ^ 5 1 - + * 4 2 - 3 * 5 + 4 2 - 3 * 5 + 3 4 2
+               "3 4 2 * 1 3 - 2 ^ / + 9 3 4 + * 6 / 1 - 8 2 ^ 5 1 - + * 1 2 + 4 * 3 + 4 2 - 3 * 6
+               "4 2 - 3 * 5 + 9 3 4 + * 6 / 1 - 8 2 ^ 5 1 - + * 4 2 - 3 * 5 + 4 2 - 3 * 5 + 3 4 2
+               "3 4 2 * 1 3 - 2 ^ / + 9 3 4 + * 6 / 1 - 8 2 ^ 5 1 - + * 1 2 + 4 * 3 + 4 2 - 3 * 6
+               "4 2 - 3 * 5 + 9 3 4 + * 6 / 1 - 8 2 ^ 5 1 - + * 4 2 - 3 * 5 + 4 2 - 3 * 5 + 3 4 2
+               "3 4 2 * 1 3 - 2 ^ / + 9 3 4 + * 6 / 1 - 8 2 ^ 5 1 - + * 1 2 + 4 * 3 + 4 2 - 3 * 6
+               "4 2 - 3 * 5 + 9 3 4 + * 6 / 1 - 8 2 ^ 5 1 - + * 4 2 - 3 * 5 + 4 2 - 3 * 5 + 3 4 2
+       }
+   l := len(postfixes);
+       for i := 0; i < l; i++ {
+               b.Run(fmt.Sprintf("len=%d", len(postfixes[i])), func(b *testing.B) {
+                       CalculatePostfix(postfixes[i])
+               })
+       }
+ }
```

Результати виконання:

```
goos: linux
goarch: amd64
pkg: github.com/Destaby/architecture-lab-2
cpu: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
BenchmarkCalculatePostfix/len=13        1000000000        0.0000012 ns/op
BenchmarkCalculatePostfix/len=33        1000000000        0.0000021 ns/op
BenchmarkCalculatePostfix/len=49        1000000000        0.0000028 ns/op
BenchmarkCalculatePostfix/len=55        1000000000        0.0000032 ns/op
BenchmarkCalculatePostfix/len=75        1000000000        0.0000038 ns/op
BenchmarkCalculatePostfix/len=97        1000000000        0.0000050 ns/op
BenchmarkCalculatePostfix/len=105       1000000000        0.0000051 ns/op
BenchmarkCalculatePostfix/len=131       1000000000        0.0000101 ns/op
BenchmarkCalculatePostfix/len=203       1000000000        0.0000129 ns/op
BenchmarkCalculatePostfix/len=335       1000000000        0.0000149 ns/op
BenchmarkCalculatePostfix/len=539       1000000000        0.0000184 ns/op
BenchmarkCalculatePostfix/len=875       1000000000        0.0000298 ns/op
BenchmarkCalculatePostfix/len=1415      1000000000        0.0000555 ns/op
BenchmarkCalculatePostfix/len=2291      1000000000        0.0000724 ns/op
BenchmarkCalculatePostfix/len=3707      1000000000        0.0001780 ns/op
BenchmarkCalculatePostfix/len=5999      1000000000        0.0002705 ns/op
BenchmarkCalculatePostfix/len=9707      1000000000        0.0003687 ns/op
PASS
ok      github.com/Destaby/architecture-lab-2   0.036s
```

Частина друга

Для 3-тої роботи ми побудували наступну діаграму взаємодії компонентів у нашій імплементації:



Система підтримки управління віртуальними машинами та балансерами навантаження. Користувач може ініціалізувати кілька машин, які виконують однакову функцію та підключити їх до балансера, який рівномірно розподілятиме запити до цих машин. Машина може бути помічена як неробоча. У такому випадку, балансер повинен перестати відправляти на неї запити.

Клієнт надсилає запити до серверу (machines/balancers). У цей час сервер обробляє його та надсилає сигнал для оновлення статусу машини та список балансирів. Store зв'язується з базою даних та надсилає SQL запити.

Частина третя

Для 4-ої роботи функція для підтвердження лінійного часу роботи парсера команд:

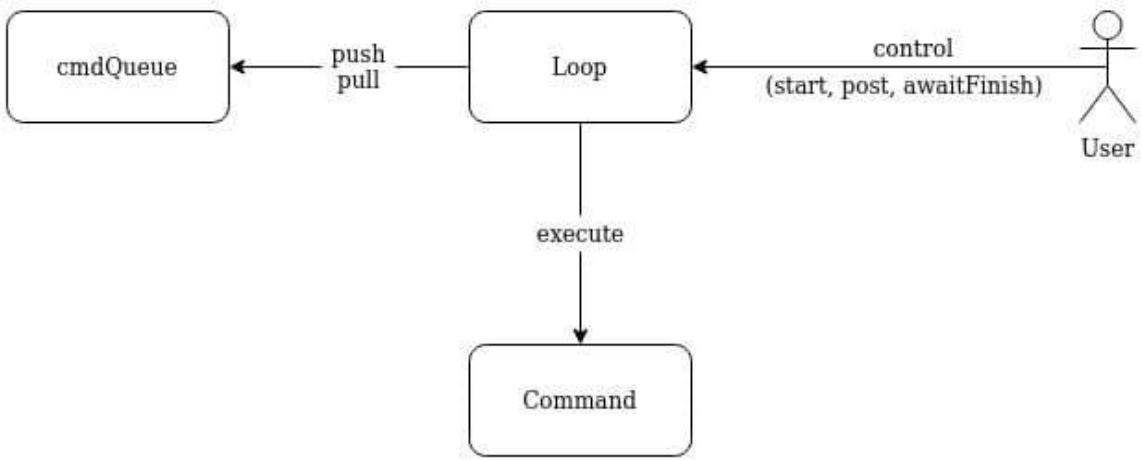
```go
package main

import (
        "testing"
        "fmt"
        "strings"
)

func BenchmarkParse(b *testing.B) {
        commands := []string{
                "print",
                "palindrom",
        }
        const baseLen = 5

        for i := 0; i < 20; i++ {
                l := baseLen * 1<<i
                in := make([]string, l)
                in[0] = commands[i % 2]
                for k := 1; k < l; k++ {
                 in[k] = "a"
                }
                b.Run(fmt.Sprintf("len=%d", l), func(b *testing.B) {
                  parse(strings.Join(in, ""))
                })
         }
}
```

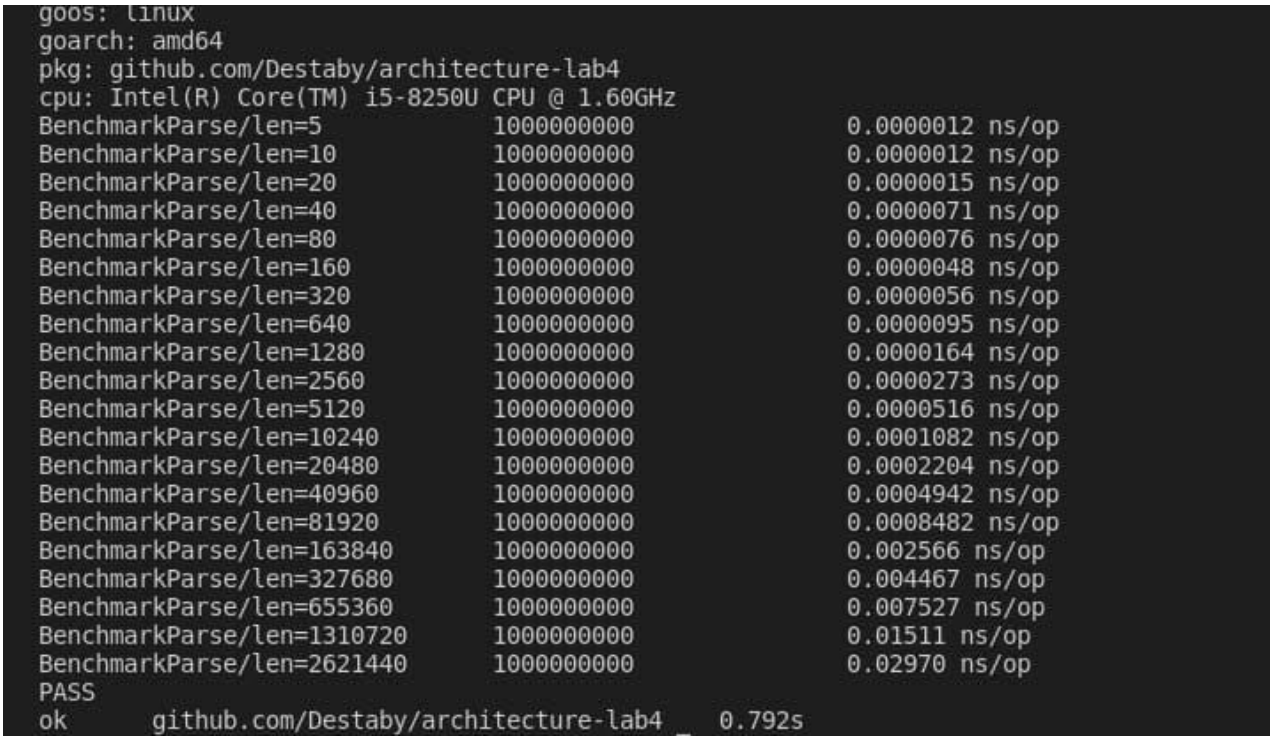
Побудована діаграма взаємодії для нашої реалізації:



Інтерпретатор читатє вхідний файл рядок за рядком, парсить команду в прочитаному рядку та додаватє команду в чергу обробки циклу подій. На

початку програми створюється екземпляр EventLoop у пакеті engine. Він має 3 публічних метода: Start, який створює чергу повідомлень - messageQueue та запускає цикл зчитування команд з даної черги, Post(Command) (який додає нову команду до черги для обробки) та AwaitFinish (який очікує, поки всі заплановані команди, що було додано до черги повідомлень, завершаться).
У даному інтерфейсі є лише один метод Execute, який виконується всередині циклу подій. Його аргумент надає імплементації команди можливість планувати виконання додаткових команд.
Далі EventLoop виводить послідовний результат (push/pull).

Результати виконання функції виміру часу:



Висновки:
При виконанні розрахунково-графічної роботи ми змогли закріпити навички ілюстрації програмних систем та оцінки часу виконання алгоритмів.
Розробили бенчмарки для двух робіт, які виконували раніше, та діаграми взаємодії.