МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

Звіт

З лабораторної роботи № 4 з дисципліни
«Моделювання систем»

«Оцінка точності та складності алгоритму імітації»

Виконав(ла) *ІП-13 Бабіч Денис* _____

(шифр, прізвище, ім'я, по батькові)

Перевірив(ла) *Дифучин А. Ю.* _____

(посада, прізвище, ім'я, по батькові)

Київ 2024

ОСНОВНА ЧАСТИНА

Мета роботи: Побудувати імітаційні моделі системи з використанням формалізму моделі масового обслуговування.

1. Розробити модель масового обслуговування, яка складається з N систем масового обслуговування. Число N є параметром моделі. Кількість подій в моделі оцінюється числом N+1. **20 балів.**
2. Виконати експериментальну оцінку складності алгоритму імітації мережі масового обслуговування. Для цього виконайте серію експериментів, в якій спостерігається збільшення часу обчислення алгоритму імітації при збільшенні кількості подій в моделі. **40 балів.**
3. Виконати теоретичну оцінку складності побудованого алгоритму імітації. **30 балів.**
4. Повторіть експеримент при зміні структури мережі масового обслуговування. **10 балів.**

Рисунок 1.1 – Завдання лабораторного практикуму

Виконання завдання:

Було виконано 10 прогонів бенчмарку для кількості подій від 100 до 1000, з кроком 100 та з використанням експоненційного генератор з математичним очікуванням 1.

```
|LOG| [BENCHMARK] Iteration #1

|LOG| [BENCHMARK] n: 100; Duration: 69ms
|LOG| [BENCHMARK] n: 200; Duration: 185ms
|LOG| [BENCHMARK] n: 300; Duration: 286ms
|LOG| [BENCHMARK] n: 400; Duration: 400ms
|LOG| [BENCHMARK] n: 500; Duration: 507ms
|LOG| [BENCHMARK] n: 600; Duration: 554ms
|LOG| [BENCHMARK] n: 700; Duration: 681ms
|LOG| [BENCHMARK] n: 800; Duration: 844ms
|LOG| [BENCHMARK] n: 900; Duration: 868ms
|LOG| [BENCHMARK] n: 1000; Duration: 963ms
```

Рисунок 1.2 – Приклад виводу для кожної ітерації системи послідовних моделей

```
|REPORT| [BENCHMARK] n: 100; Duration (mean): 64ms
|REPORT| [BENCHMARK] n: 200; Duration (mean): 176.7ms
|REPORT| [BENCHMARK] n: 300; Duration (mean): 283.4ms
|REPORT| [BENCHMARK] n: 400; Duration (mean): 383.8ms
|REPORT| [BENCHMARK] n: 500; Duration (mean): 468.9ms
|REPORT| [BENCHMARK] n: 600; Duration (mean): 560.9ms
|REPORT| [BENCHMARK] n: 700; Duration (mean): 653.3ms
|REPORT| [BENCHMARK] n: 800; Duration (mean): 752.7ms
|REPORT| [BENCHMARK] n: 900; Duration (mean): 857.4ms
|REPORT| [BENCHMARK] n: 1000; Duration (mean): 951ms
```

Рисунок 1.3 – Результат роботи бенчмарку системи послідовних моделей

Кількість подій	Час виконання, мс
100	69
200	183
300	306
400	399
500	490
600	589
700	690
800	787
900	890
1000	971

Рисунок 1.4 – Метрики для систем з послідовних моделей

Рисунок 1.5 – Візуалізація метриків систем з послідовних моделей для 10 ітерацій

$$O(v \cdot timeMod \cdot k)$$

Де v – інтенсивність подій (значення), *timeMod* – час моделювання, k – кількість елементарних операцій для обробки однієї події, що вираховується як кількість успішних операцій за одиницю часу ($\frac{EOPs}{S}$).

Кількість подій	Час виконання, мс	v	k	timeMod	O
100	69	101	26	500	1313000
200	183	201	26	500	2613000
300	306	301	26	500	3913000
400	399	401	26	500	5213000
500	490	501	26	500	6513000
600	589	601	26	500	7813000
700	690	701	26	500	9113000
800	787	801	26	500	10413000
900	890	901	26	500	11713000
1000	971	1001	26	500	13013000

Рисунок 1.6 – Підраховані теоретичні значення складності системи

```
|LOG| [BENCHMARK] Iteration #1

|LOG| [BENCHMARK] n: 100; Duration: 12ms
|LOG| [BENCHMARK] n: 200; Duration: 44ms
|LOG| [BENCHMARK] n: 300; Duration: 98ms
|LOG| [BENCHMARK] n: 400; Duration: 161ms
|LOG| [BENCHMARK] n: 500; Duration: 233ms
|LOG| [BENCHMARK] n: 600; Duration: 342ms
|LOG| [BENCHMARK] n: 700; Duration: 437ms
|LOG| [BENCHMARK] n: 800; Duration: 603ms
|LOG| [BENCHMARK] n: 900; Duration: 717ms
|LOG| [BENCHMARK] n: 1000; Duration: 892ms
```

Рисунок 1.7 – Приклад виводу для кожної ітерації системи паралельних моделей

```
|REPORT| [BENCHMARK] n: 100; Duration (mean): 11.4ms
|REPORT| [BENCHMARK] n: 200; Duration (mean): 41.8ms
|REPORT| [BENCHMARK] n: 300; Duration (mean): 90.7ms
|REPORT| [BENCHMARK] n: 400; Duration (mean): 156.7ms
|REPORT| [BENCHMARK] n: 500; Duration (mean): 236.2ms
|REPORT| [BENCHMARK] n: 600; Duration (mean): 336.2ms
|REPORT| [BENCHMARK] n: 700; Duration (mean): 442.6ms
|REPORT| [BENCHMARK] n: 800; Duration (mean): 582.6ms
|REPORT| [BENCHMARK] n: 900; Duration (mean): 758.1ms
|REPORT| [BENCHMARK] n: 1000; Duration (mean): 915.2ms
```

Рисунок 1.8 – Результат роботи бенчмарку системи паралельних моделей

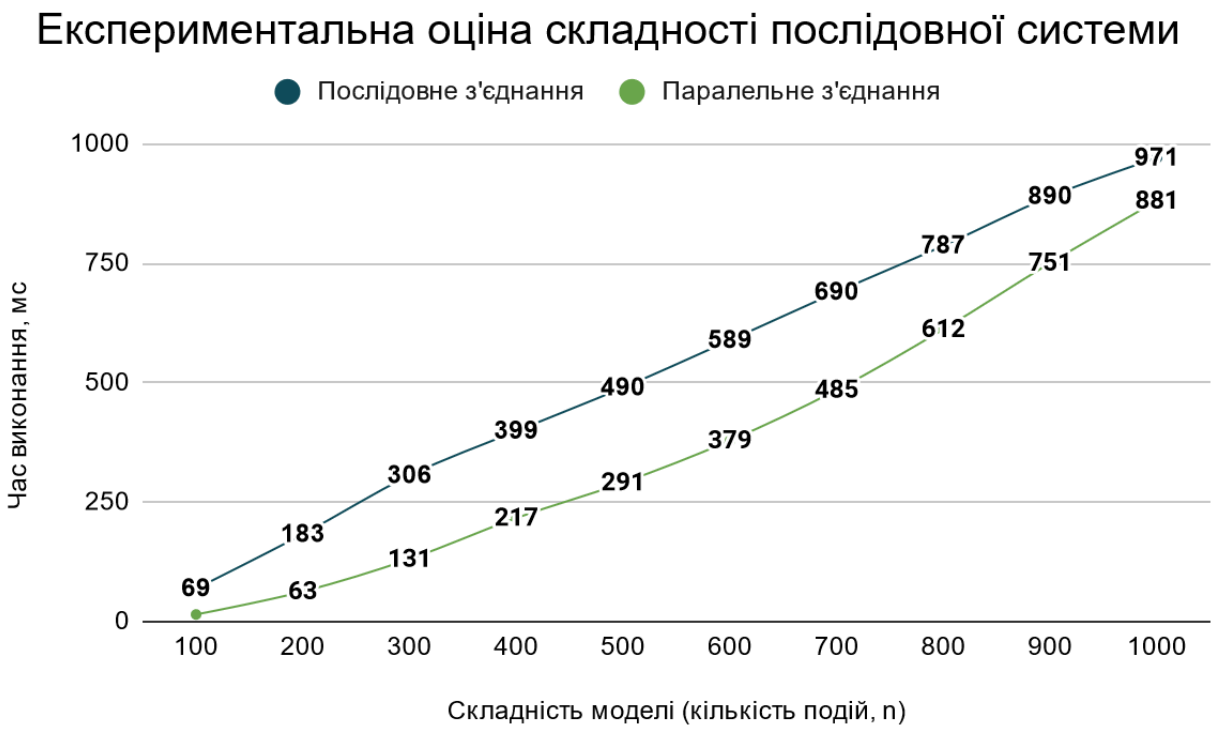

Рисунок 1.9 – Порівняння результатів паралельної та послідовної моделі

ВИСНОВКИ

У ході виконання лабораторної роботи було створено імітаційні моделі систем масового обслуговування, що дозволило дослідити їх продуктивність та ефективність у послідовному й паралельному виконанні. Було проведено серію експериментів, що охоплюють 10 прогонів бенчмарку з різною кількістю подій від 100 до 1000 із кроком 100, використовуючи експоненційний генератор з математичним очікуванням 1. Результати кожного прогону зафіксовані у вигляді вихідних даних, які демонструють поведінку системи при обробці різного навантаження.

Метрики, отримані в процесі експериментів, дозволили оцінити продуктивність системи для послідовних моделей. Зокрема, було розраховано кількість елементарних операцій на одну подію (EOPs) та складність системи у вигляді формули $O(v \cdot timeMod \cdot k)$. Для кожної ітерації виконано візуалізацію метрик, що дозволило визначити залежності між параметрами системи та її продуктивністю. Теоретичні значення складності підтвердили відповідність експериментальних даних розрахунковим прогнозам.

Окрім цього, було проаналізовано роботу паралельних моделей системи, результати яких порівняно з послідовними моделями. Порівняння показало, що паралельні моделі демонструють вищу продуктивність при великій інтенсивності подій завдяки можливості одночасної обробки кількох різних задач паралельно.

ДОДАТОК ПРОГРАМНИЙ КОД

Programs.cs

```
using System;
using System.Collections.Generic;
using LabWork4.Framework.Core.Controllers;
using LabWork4.Framework.Components.Queues.Concrete;
using LabWork4.Framework.Components.Modules.Common;
using LabWork4.Framework.Components.Modules.Concrete;
using LabWork4.Framework.Components.Workers.Concrete;
using LabWork4.Framework.Components.Schemes.Concrete;
using LabWork4.Framework.Components.Tasks.Utilities.Factories.Concrete;

namespace LabWork4.Application;

file sealed class Program
{
    private static void Main()
    {
        const float SIMULATION_TIME = 500.0f;
        const int BENCHMARK_MILLISECONDS = 1000;

        const int ITERATIONS_COUNT = 10;

        const int LINES_COUNT = 10;
        const int MODULES_COUNT_START = 100;
        const int MODULES_COUNT_DELTA = 100;
        const int MODULES_COUNT_FINISH = 1000;
```

```
        // Program.CreateBenchmarkModel().RunSimulation(SIMULATION_TIME,
BENCHMARK_MILLISECONDS);

    IDictionary<int, IList<int>> samples = new Dictionary<int, IList<int>>();

        for (int modulesCount = MODULES_COUNT_START; modulesCount <=
MODULES_COUNT_FINISH; modulesCount += MODULES_COUNT_DELTA)
        samples[modulesCount] = new int[ITERATIONS_COUNT];

    for (int iteration = 0; iteration < ITERATIONS_COUNT; ++iteration)
    {
            Console.WriteLine($"\n|LOG| [BENCHMARK] Iteration #{iteration +
1}\n");

            for (int modulesCount = MODULES_COUNT_START; modulesCount <=
MODULES_COUNT_FINISH; modulesCount += MODULES_COUNT_DELTA)
        {
                                        SimulationModelController model =
Program.CreateSequentialModel(modulesCount);
                                    // SimulationModelController model =
Program.CreateParallelModel(modulesCount, LINES_COUNT);

            model.RunSimulation(SIMULATION_TIME);
                                            samples[modulesCount][iteration] =
model.SimulationDurationMilliseconds;
                Console.WriteLine($"|LOG| [BENCHMARK] n: {modulesCount};
Duration: {model.SimulationDurationMilliseconds}ms");
        }
    }
```

```
        for (int modulesCount = MODULES_COUNT_START; modulesCount <= MODULES_COUNT_FINISH; modulesCount += MODULES_COUNT_DELTA)
            Console.Write($"\n|REPORT| [BENCHMARK] n: {modulesCount}; Duration (mean): {Program.CalculateDurationMean(samples[modulesCount])}ms");
    }

    private static BenchmarkModelController CreateBenchmarkModel()
    {
        DisposeModule dispose = new DisposeModule("dispose");
        ProcessorModule processor = new ProcessorModule("processor", new SingleTransitionScheme(dispose), new MockExponentialWorker(1.0f), new DefaultQueue(Int32.MaxValue));
        CreateModule create = new CreateModule("create", new SingleTransitionScheme(processor), new MockExponentialWorker(1.0f), new MockTaskFactory());
        return new BenchmarkModelController(new Module[] { create, processor, dispose });

    }

    private static SimulationModelController CreateSequentialModel(int modelsCount)
    {
        ProcessorModule nextProcessor;
        ProcessorModule previousProcessor;
        IList<Module> modules = new List<Module>();

        DisposeModule dispose = new DisposeModule("dispose");
```

```
        modules.Add((Module)dispose);

         previousProcessor = new ProcessorModule($"processor_{modelsCount}", new
SingleTransitionScheme(dispose), new MockExponentialWorker(1.0f), new
DefaultQueue(Int32.MaxValue));
        modules.Add((Module)previousProcessor);

        for (int i = modelsCount - 1; i > 0; --i)
        {
                        nextProcessor = new ProcessorModule($"processor_{i}", new
SingleTransitionScheme(previousProcessor), new MockExponentialWorker(1.0f),
new DefaultQueue(Int32.MaxValue));
           modules.Add((Module)nextProcessor);
           previousProcessor = nextProcessor;
        }

                                    modules.Add(new CreateModule("create", new
SingleTransitionScheme(previousProcessor), new MockExponentialWorker(1.0f),
new MockTaskFactory()));

        return new SimulationModelController(modules);
    }

     private static SimulationModelController CreateParallelModel(int modelsCount,
int linesCount)
    {
        const float MAX_PROBABILITY = 1.0f;
        const int DIRECT_CREATE_CONNECTIONS_COUNT = 1;
```

```
float flowProbability = MAX_PROBABILITY / linesCount;
                int modelsPerLineCount = (modelsCount / linesCount) -
DIRECT_CREATE_CONNECTIONS_COUNT;

ProcessorModule nextProcessor;
ProcessorModule previousProcessor;
IList<Module> modules = new List<Module>();

DisposeModule dispose = new DisposeModule("dispose");
modules.Add((Module)dispose);

ProbabilityScheme createScheme = new ProbabilityScheme(dispose);

for (int i = 0; i < linesCount; ++i)
{
        previousProcessor = new ProcessorModule($"processor_{i}_{0}", new
SingleTransitionScheme(dispose), new MockExponentialWorker(1.0f), new
DefaultQueue(Int32.MaxValue));
    modules.Add((Module)previousProcessor);

    for (int j = modelsPerLineCount - 1; j >= 0; --j)
    {
            nextProcessor = new ProcessorModule($"processor_{i}_{j}", new
SingleTransitionScheme(previousProcessor), new MockExponentialWorker(1.0f),
new DefaultQueue(Int32.MaxValue));
        modules.Add((Module)nextProcessor);
        previousProcessor = nextProcessor;
    }
```

```
            createScheme.Attach(previousProcessor, flowProbability);
        }

                    modules.Add(new CreateModule("create", createScheme, new
MockExponentialWorker(1.0f), new MockTaskFactory()));

        return new SimulationModelController(modules);
    }

    private static float CalculateDurationMean(IList<int> samples)
    {
        float durationTotal = 0.0f;

        for (int i = 0; i < samples.Count; ++i)
            durationTotal += samples[i];

        return durationTotal / samples.Count;
    }
}
```

BenchmarkModelController.cs

```
using System;
using System.Linq;
using System.Diagnostics;
using System.Collections.Generic;
using LabWork4.Framework.Common;
using LabWork4.Framework.Components.Modules.Common;

namespace LabWork4.Framework.Core.Controllers;
```

```
internal sealed class BenchmarkModelController : IStatisticsPrinter
{
    private readonly Stopwatch stopwatch;
    private readonly IList<Module> modules;

    private float timeNext;
    private float timeCurrent;

    internal BenchmarkModelController(IList<Module> modules)
    {
        if (modules == null)
            throw new ArgumentNullException($"{nameof(modules)} cannot be null.");

        this.stopwatch = new Stopwatch();

        this.timeNext = 0.0f;
        this.modules = modules;
        this.timeCurrent = 0.0f;
    }

                    internal int SimulationDurationMilliseconds =>
this.stopwatch.Elapsed.Milliseconds;

    internal void RunSimulation(float simulationTime, int milliseconds)
    {
        IList<Module> nextModules;
        this.timeNext = this.modules.Min(module => module.TimeNext);
```

```
        this.stopwatch.Restart();

        while (timeNext < simulationTime)
        {
            if (this.stopwatch.ElapsedMilliseconds >= milliseconds)
                break;

            this.timeCurrent = this.timeNext;

            foreach (Module module in this.modules)
                module.TimeCurrent = this.timeCurrent;

                    nextModules = this.modules.Where(module => module.TimeNext ==
this.timeCurrent).ToList();

            foreach (Module module in nextModules)
                module.CompleteTask();

            this.timeNext = this.modules.Min(module => module.TimeNext);

            this.PrintIntermediateStatistics();
        }

        this.stopwatch.Stop();

        this.PrintFinalStatistics();
    }

    public void PrintIntermediateStatistics()
```

```
    {
        foreach (Module module in this.modules)
            module.PrintIntermediateStatistics();
    }

    public void PrintFinalStatistics()
    {
        foreach (Module module in this.modules)
            module.PrintFinalStatistics();
    }
}
```

SimulationModelController.cs

```
using System;
using System.Linq;
using System.Diagnostics;
using System.Collections.Generic;
using LabWork4.Framework.Common;
using LabWork4.Framework.Components.Modules.Common;

namespace LabWork4.Framework.Core.Controllers;

internal sealed class SimulationModelController : IStatisticsPrinter
{
    private readonly Stopwatch stopwatch;
    private readonly IList<Module> modules;

    private float timeNext;
    private float timeCurrent;
```

```
    internal SimulationModelController(IList<Module> modules)
    {
        if (modules == null)
            throw new ArgumentNullException($"{nameof(modules)} cannot be null.");

        this.stopwatch = new Stopwatch();

        this.timeNext = 0.0f;
        this.modules = modules;
        this.timeCurrent = 0.0f;
    }

                    internal int SimulationDurationMilliseconds =>
this.stopwatch.Elapsed.Milliseconds;

    internal void RunSimulation(float simulationTime)
    {
        IList<Module> nextModules;
        this.timeNext = this.modules.Min(module => module.TimeNext);

        this.stopwatch.Restart();

        while (timeNext < simulationTime)
        {
            this.timeCurrent = this.timeNext;

            foreach (Module module in this.modules)
                module.TimeCurrent = this.timeCurrent;
```

```
                nextModules = this.modules.Where(module => module.TimeNext ==
this.timeCurrent).ToList();

            foreach (Module module in nextModules)
                module.CompleteTask();

            this.timeNext = this.modules.Min(module => module.TimeNext);

            // this.PrintIntermediateStatistics();
        }

        this.stopwatch.Stop();

        // this.PrintFinalStatistics();
    }

    public void PrintIntermediateStatistics()
    {
        foreach (Module module in this.modules)
            module.PrintIntermediateStatistics();
    }

    public void PrintFinalStatistics()
    {
        foreach (Module module in this.modules)
            module.PrintFinalStatistics();
```

```
                // Console.WriteLine($"|LOG| [SYSTEM] Duration: {this.stopwatch.Elapsed}");
    }
}
```