НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Звіт по лабораторній роботі №4
«Моделювання систем»
«Оцінка точності та складності алгоритму імітації»

Студент: Галько М.В.
Група: ІП-01

Київ, 2023

ЗМІСТ

Завдання

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**

1. Побудова послідовної моделі

Розробимо статичний клас ModelHelper із методом CreateSequential. В нього необхідно передавати кількість процесів яка має бути у фінальній моделі:

```
public static Model CreateSequential(int processCount)
{
  List<Element> elements = new() { new Create() };
  for (int n = 0; n < processCount; n++)
  {
    elements.Add(new Process());
    elements[^2].NextElementsContainer = new NextElementContainer(elements[^1]);
  }
  return new Model(elements);
}
```

2. Результати експериментів послідовної моделі

2.1 Програма

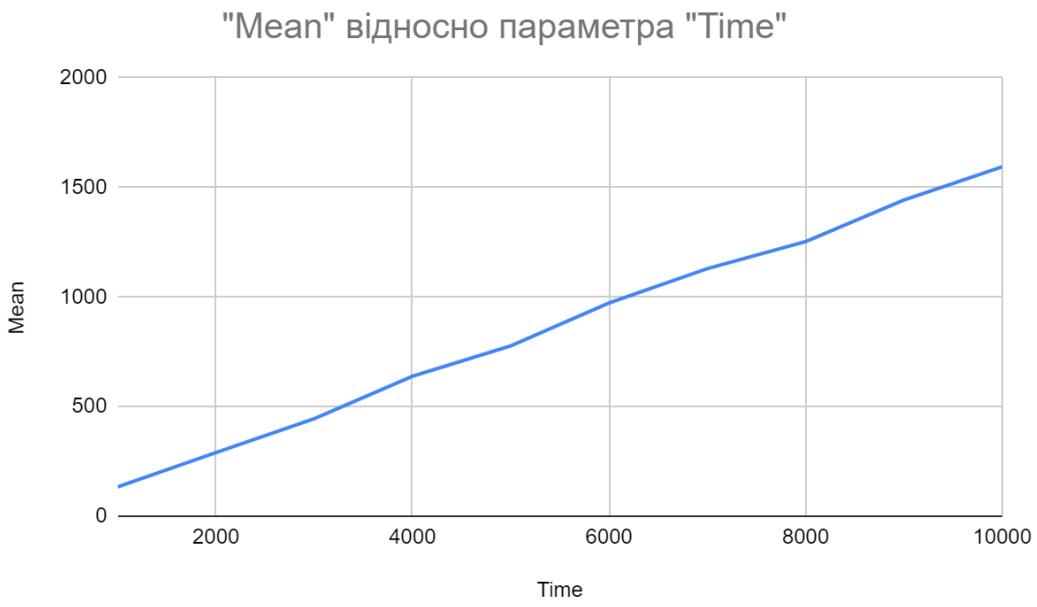
Для оцінки складності алгоритму будемо використовувати готовий інструмент BenchmarkDotNet. Виконаємо тести продуктивності для моделі з 50-ти послідовних процесів із часом симуляції від 1000 до 10000 із кроком 1000. Отже виконаємо 10 тестів. Для цього сформуємо наступний код:

```
BenchmarkRunner.Run<Lab4Benchmark>();
public class Lab4Benchmark
{
    [Params(1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000)]
    public int ModelTime;
    private Model? _sequentialModel;

    [IterationSetup]
    public void CreateModel()
    {
        _sequentialModel = ModelHelper.CreateSequential(50);
    }
    [Benchmark]
    public void SimulateSequential() => _sequentialModel.Simulate(time: ModelTime,
printResult: false);
}
```

2.2 Результати

Method	ModelTime	Mean	Error	StdDev	Median
Simulate	1000	133.4 ms	2.65 ms	3.06 ms	133.5 ms
Simulate	2000	289.7 ms	3.08 ms	2.89 ms	289.3 ms
Simulate	3000	444.5 ms	5.74 ms	5.37 ms	442.9 ms
Simulate	4000	638.6 ms	12.12 ms	22.17 ms	633.7 ms
Simulate	5000	776.5 ms	6.38 ms	5.66 ms	776.5 ms
Simulate	6000	973.1 ms	18.56 ms	48.57 ms	966.3 ms
Simulate	7000	1,129.1 ms	22.43 ms	61.79 ms	1,111.3 ms
Simulate	8000	1,252.2 ms	11.99 ms	10.63 ms	1,252.1 ms
Simulate	9000	1,442.0 ms	26.19 ms	42.30 ms	1,436.1 ms
Simulate	10000	1,593.5 ms	25.13 ms	22.28 ms	1,596.9 ms



3. Теоретична оцінка складності

Теоретична оцінка складності виконується за формулою O(v*timeMod*k), де v – інтенсивність подій (delay для елемента Create = 1), timeMode – час моделювання (ModelTime), k – середня кількість елементарних операцій для обробки 1-єї події (кількість елементів Process = 50).

Отже, наша формула змінює вигляд на O(1 * ModelTime * N) = O(N), що є лінійною залежністю. Бачимо, що результативний графік відповідає цій умові.

4. Приклад на розгалуженої моделі

4.1 Код розгалуженої моделі

```
public static Model CreateBranching2Levels(int processCount, int nextProcessCount)
{
  List<Element> allElements = new() { new Create() };
  NextElementsContainerByProbability createContainer = new();
  allElements[^1].NextElementsContainer = createContainer;
  for (int n = 0; n < processCount; n++) {
    allElements.Add(new Process());
    createContainer.AddNextElement(allElements[^1], 1.0 / processCount);
    NextElementsContainerByProbability processContainer = new();
    allElements[^1].NextElementsContainer = processContainer;
    for (int m = 0; m < nextProcessCount; m++){
      allElements.Add(new Process(name: "  PROCESS_" + (n+1) + "_" + m));
      processContainer.AddNextElement(allElements[^1], 1.0 / nextProcessCount);
    }
  }
  return new Model(allElements);
}
```

4.1 Результати

Виконавши ідентичне застосування інструмента BenchmarkDotNet для розгалуженої моделі (структура: 1 Create, 10 Process та від кожного Process йде 5 Process) отримуємо наступний результат:

Method	ModelTime	Mean	Error	StdDev	Median
SimulateBranching	1000	16.11 ms	0.927 ms	2.599 ms	15.57 ms
SimulateBranching	2000	29.38 ms	0.711 ms	1.992 ms	29.12 ms
SimulateBranching	3000	44.05 ms	1.118 ms	3.225 ms	43.37 ms
SimulateBranching	4000	58.87 ms	1.810 ms	5.251 ms	57.63 ms
SimulateBranching	5000	71.71 ms	1.766 ms	5.208 ms	70.46 ms
SimulateBranching	6000	83.66 ms	1.663 ms	4.876 ms	82.66 ms
SimulateBranching	7000	96.80 ms	1.915 ms	4.625 ms	95.86 ms
SimulateBranching	8000	115.05 ms	4.321 ms	12.116 ms	111.49 ms
SimulateBranching	9000	123.80 ms	2.454 ms	5.540 ms	122.67 ms
SimulateBranching	10000	137.04 ms	2.718 ms	6.298 ms	136.30 ms

Висновок

В ході виконання лабораторної роботи було створено статичний клас ModelHelper який реалізує 2 методи CreateSequential та CreateBranch2Levels, які повертають послідовну та розгалужену глубини 2 моделі.

За допомогою інструмента BenchmarkDotNet були протестовані 2 моделі із різним часом симуляції від 1000 до 10000 із кроком 1000 та підрахована їх теоретична оцінка складності. Проаналізувавши функцію оцінки складності і її параметрів отримали лінійну закономірність O(N), де N – кількість елементів типу Process. Перевівши табличні результати BenchmarkDotNet у графічний вигляд, продемонстрували відношення Mean до ModelTime. З результатів графіків видна лінійна закономірність для обох моделей.

Для побудови моделей були використані авторські бібліотеки сформовані під час виконання 1-3 лабораторних робіт:

Код бібліотек