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

## Факультет інформатики та обчислювальної техніки

## Кафедра інформатики та програмної інженерії

## Звіт по лабораторній роботі №3

## «Моделювання систем»

## «Побудова імітаційної моделі системи з використанням формалізму моделі масового обслуговування»

Студент: Галько М.В. і

Група: ІП-01 і

## Київ, 2023

# ЗМІСТ

# 

[**Завдання 1**](#_fjx4n96j1zd4) **2**

[Завдання](#_a6r9ya1xrwiu) 2

[Діаграма структури](#_4h8l6zz4bvs7) 2

[Код побудови системи](#_uuqc50b5scp0) 2

[Вивід процесу](#_ibgx4iol9moq) 3

[Результат](#_ewy3g65wiewv) 4

[**Завдання 2**](#_k2wjm7outw88) **6**

[Завдання](#_nyd8thl445xp) 6

[Діаграма структури](#_4ghjetcdok35) 7

[Додаткові умови та їх реалізація](#_6vt7k0vm60dj) 7

[Код побудови системи](#_8zdzxpgos8pg) 9

[Вивід процесу](#_2cefk6vaozwl) 9

[Результат](#_qgyzwvc74e87) 11

[**Завдання 3**](#_u78md1wkzlq4) **13**

[Завдання](#_hc9rdo5vhbo8) 13

[Діаграма структури](#_r5vtjndgoamc) 15

[Додаткові умови та їх реалізація](#_mxkcnstdq3bd) 15

[Код побудови системи](#_g1tnql6jwysu) 18

[Результат](#_4d3zb4j8s4n1) 23

[Результати нової моделі](#_tamx1e81qx7q) 25

# 

# 

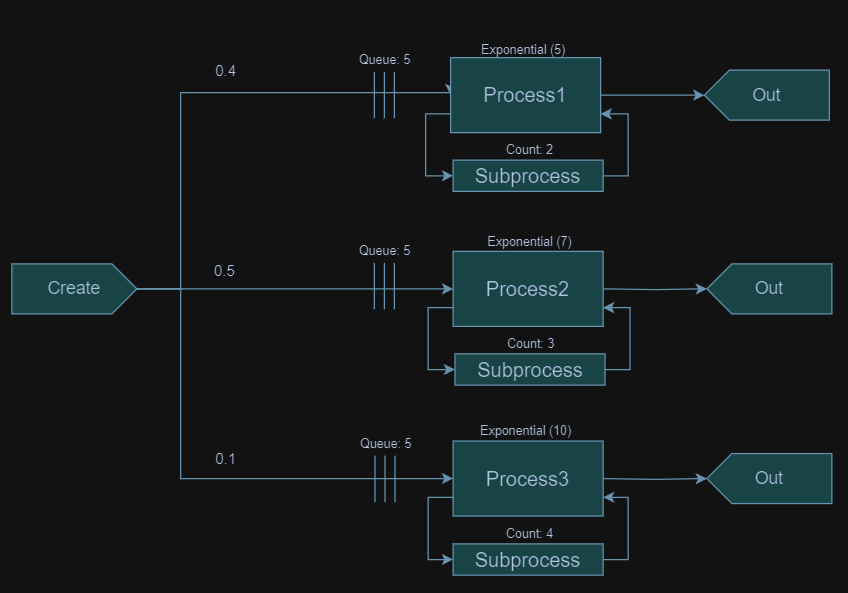
# Завдання 1

## Завдання

Реалізувати універсальний алгоритм імітації моделі масового обслуговування з багатоканальним обслуговуванням, з вибором маршруту за пріоритетом або за заданою ймовірністю. **30 балів.**

## Діаграма структури

Реалізуємо систему із Create та 3-а Process елементами:



## Код побудови системи

public class Task1

{

public Model Model;

private Create \_create = new();

private Process \_process1 = new(5, 2, maxQueue: 5);

private Process \_process2 = new(7, 3, maxQueue: 5);

private Process \_process3 = new(10, 4, maxQueue: 5);

public Task1()

{

var container = new NextElementsContainerByProbability();

container.AddNextElement(\_process1, 40);

container.AddNextElement(\_process2, 50);

container.AddNextElement(\_process3, 10);

\_create.NextElementsContainer = container;

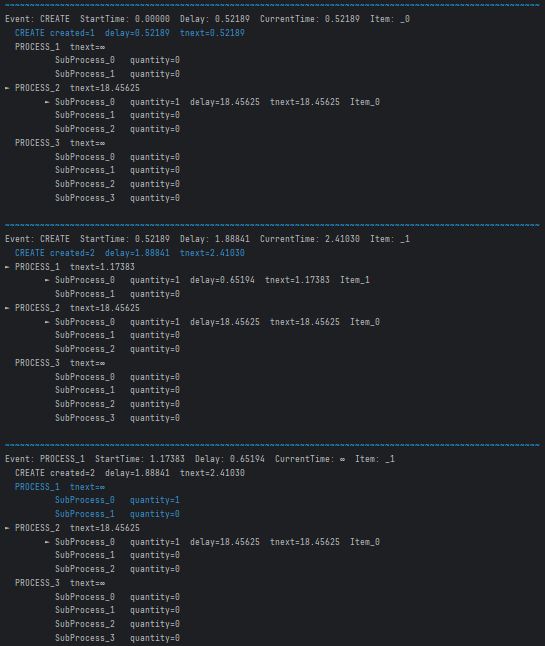
Model = new Model(new List<Element> { \_create, \_process1, \_process2, \_process3 });

}

}

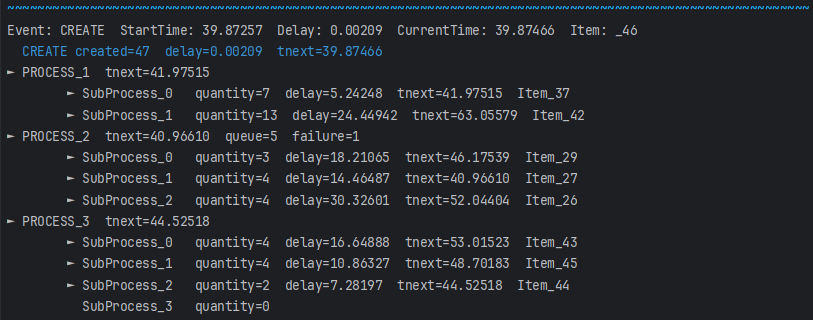
## Вивід процесу

Запустимо симуляцію на час 100. Отже маємо наступний вивід:



У перших 2-х кроках створюються об’єкти. Item\_0 надходить у Process\_2, а Item\_1 у Process\_1. У 3-ому кроці Process\_1 завершує обробку Item\_1 у SubProcess\_0.

Приклад отримання помилки:



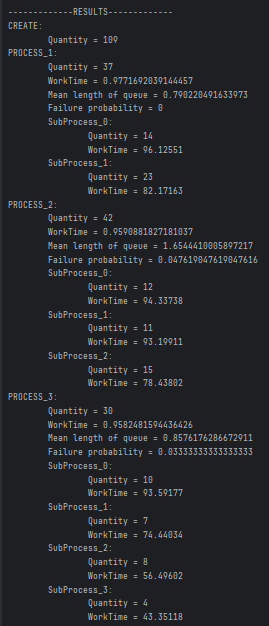
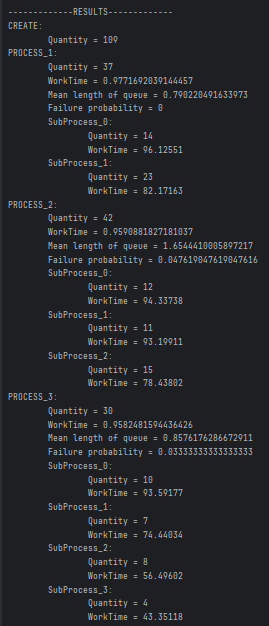
Оскільки максимальний розмір черги 5, то при створенні об’єкту, який потрапив до Process\_2, він не може потрапити у чергу (повністю зайнята).

## Результат

CREATE: Quantity = 104

| PROCESS\_1:  Quantity = 42  WorkTime = 0.895  Mean queue = 0.704  Failure prob. = 0    SubProcess\_0:  Quantity = 23  WorkTime = 75.605    SubProcess\_1:  Quantity = 18  WorkTime = 75.302 | PROCESS\_2:  Quantity = 45  WorkTime = 1  Mean queue = 1.982  Failure prob. = 0.178  SubProcess\_0:  Quantity = 9  WorkTime = 93.325  SubProcess\_1:  Quantity = 12  WorkTime = 93.164  SubProcess\_2:  Quantity = 12  WorkTime = 81.19 | PROCESS\_3:  Quantity = 17  WorkTime = 0.832  Mean queue = 0  Failure prob. = 0  SubProcess\_0:  Quantity = 5  WorkTime = 78.997  SubProcess\_1:  Quantity = 9  WorkTime = 48.767  SubProcess\_2:  Quantity = 3  WorkTime = 18.122  SubProcess\_3:  Quantity = 0  WorkTime = 0.00000 |
| --- | --- | --- |

Як видно з результатів, перехід за ймовірністю працює коректно. Найбільш завантажений Process\_2(50%), потім Process\_1(40%) та Process\_3(10%). Середня черга максимальна у найзавантаженішого Process\_2, проте досягає лише 2 об’єктів. Помилки трапляються лише у ньому. Для покращення роботи моделі необхідно знизити завантаженість елемента, передавши його роботу найнезавантаженим. В нашому випадку таким елементом є Process\_3, в якому SubProcess\_2 та Subprocess\_3 майже не працюють. Можна зменшити ймовірність переходу до Process\_2 до 45% і збільшити у Process\_3 до 20%, а Process\_1 до 35%. Результати:



Як видно з результатів завантаженість кожного підпроцесу стала більш рівномірною. Помилки трапляються, оскільки процес створення нових об’єктів є надто частим, але тепер вони рівномірні вцілому.

# Завдання 2

## Завдання

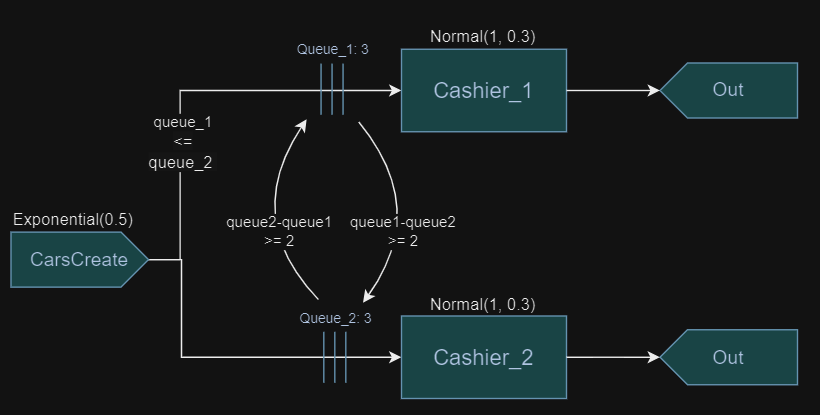
Для наступного тексту задачі скласти формалізовану модель масового обслуговування та реалізувати її з використанням побудованого універсального алгоритму. **30 балів**:

У банку для автомобілістів є два віконця, кожне з яких обслуговується одним касиром і має окрему під'їзну смугу. Обидві смуги розташовані поруч. З попередніх спостережень відомо, що інтервали часу між прибуттям клієнтів у годину пік розподілені експоненційно з математичним очікуванням, рівним 0,5 од. часу. Через те, що банк буває переобтяжений тільки в годину пік, то аналізується тільки цей період. Тривалість обслуговування в обох касирів однакова і розподілена експоненційно з математичним очікуванням, рівним 0,3 од. часу. Відомо також, що при рівній довжині черг, а також при відсутності черг, клієнти віддають перевагу першій смузі. В усіх інших випадках клієнти вибирають більш коротку чергу. Після того, як клієнт в'їхав у банк, він не може залишити його, доки не буде обслугований. Проте він може перемінити чергу, якщо стоїть останнім і різниця в довжині черг при цьому складає не менше двох автомобілів. Через обмежене місце на кожній смузі може знаходитися не більш трьох автомобілів. У банку, таким чином, не може знаходитися більш восьми автомобілів, включаючи автомобілі двох клієнтів, що обслуговуються в поточний момент касиром. Якщо місце перед банком заповнено до границі, то клієнт, що прибув, вважається втраченим, тому що він відразу ж виїжджає.

Початкові умови такі: 1) обидва касири зайняті, тривалість обслуговування для кожного касира нормально розподілена з математичним очікуванням, рівним 1 од. часу, і середньоквадратичним відхиленням, рівним 0,3 од. часу; 2) прибуття першого клієнта заплановано на момент часу 0,1 од. часу; 3) у кожній черзі очікують по два автомобіля.

Визначити величини: 1) середнє завантаження кожного касира; 2) середнє число клієнтів у банку; 3) середній інтервал часу між від'їздами клієнтів від вікон; 4) середній час перебування клієнта в банку; 5) середнє число клієнтів у кожній черзі; 6) відсоток клієнтів, яким відмовлено в обслуговуванні; 7) число змін під'їзних смуг.

## Діаграма структури



## Додаткові умови та їх реалізація

Маємо реалізувати:

1. NextElementContainerByQueuePriority для обрання касиру клієнтами за кількістю машин у черзі:

public class NextElementsContainerByQueuePriority : NextElementsContainer

{

private Dictionary<Process, int> \_nextElements = new();

public void AddNextElement(Process process, int ascendingPriority) => \_nextElements.Add(process, ascendingPriority);

protected override Element GetNextElement()

{

var elementsWithMinQueueLength = GetElementsWithMinQueueLength();

return elementsWithMinQueueLength.MinBy(e => e.Value).Key;

}

private Dictionary<Process, int> GetElementsWithMinQueueLength()

{

var minQueueLength = \_nextElements.Min(e => e.Key.QueueLength);

var nextElementsWithMinQueueLength = new Dictionary<Process, int>();

foreach (var (process, priority) in \_nextElements)

if (process.QueueLength == minQueueLength)

nextElementsWithMinQueueLength.Add(process, priority);

return nextElementsWithMinQueueLength;

}

}

1. Event OnQueueChange та метод, що підписується ChangeQueueIfNecessary для зміни черги автомобіля якщо сусідня черга менша хоча б на 2 автомобілі:

private void ChangeQueueIfNecessary()

{

if (Math.*Abs*(\_cashier1.QueueLength -\_cashier2.QueueLength) < 2) return;

Console.*WriteLine*($"\nCHANGE\_QUEUE: Q1={\_cashier1.QueueLength} Q2={\_cashier2.QueueLength}!\*!\*!\*!\*!");

if (\_cashier1.QueueLength > \_cashier2.QueueLength) {

Process.*TryChangeQueueForLastItem*(\_cashier1, \_cashier2);

Console.*WriteLine*("CASHIER1 -> CASHIER2");

}

else {

Process.*TryChangeQueueForLastItem*(\_cashier2, \_cashier1);

Console.*WriteLine*("CASHIER2 -> CASHIER1");

}

Model.QueueChangeCount7++;

}

1. Початкові умови:

const double *startTime* = 0.1;

\_cars.NextT = *startTime*;

for (var i = 0; i < 3; i++)

{

\_cashier1.InAct(new Item(*startTime*));

\_cashier2.InAct(new Item(*startTime*));

}

1. Збір статистики:

public class Task2Model : Model

{

public int QueueChangeCount7;

public Task2Model(List<Element> elements) : base(elements) {}

public override void Simulate(double time, double startTime = 0, bool printSteps = false)

{

base.Simulate(time, startTime, printSteps);

Console.*WriteLine*();

Console.*WriteLine*($"1) Average workTime:"); PrintEachProcessWorkTime\_1();

Console.*WriteLine*($"2) Average clients count: {AverageItemsCount}");

Console.*WriteLine*($"3) Average time between departures: {AverageBetweenOutActs3}");

Console.*WriteLine*($"4) Average time spent by a customer in the bank: {AverageItemTimeInSystem}");

Console.*WriteLine*($"5) Average clients count in each queue:"); PrintEachProcessMeanQueue\_5();

Console.*WriteLine*($"6) Failure probability: {FailurePercent}%");

Console.*WriteLine*($"7) Queue change count: {QueueChangeCount7}");

}

private void PrintEachProcessWorkTime\_1() => Processes.ForEach(p => Console.*WriteLine*($"\t{p.Name}: {p.WorkTime / AllTime}"));

private double AverageBetweenOutActs3 => Processes.Sum(p => p.TotalTimeBetweenOutActs / p.OutActQuantity) / Processes.Count;

private void PrintEachProcessMeanQueue\_5() => Processes.ForEach(p => Console.*WriteLine*($"\t{p.Name}: {p.MeanQueueAllTime / AllTime}"));

}

## Код побудови системи

public class Task2CarBank

{

public Task2Model Model;

private Create \_cars = new(0.5, name: "Cars");

private Process \_cashier1 = new(new NormalRandomizer(1, 0.3), maxQueue: 3, name: "Cashier1");

private Process \_cashier2 = new(new NormalRandomizer(1, 0.3), maxQueue: 3, name: "Cashier2");

public Task2CarBank() {

// Connecting elements

var cashiers = new NextElementsContainerByQueuePriority();

cashiers.AddNextElement(\_cashier1, 1);

cashiers.AddNextElement(\_cashier2, 2);

\_cars.NextElementsContainer = cashiers;

// Changing queues condition

\_cashier1.OnQueueChanged += ChangeQueueIfNecessary;

\_cashier2.OnQueueChanged += ChangeQueueIfNecessary;

// Initial states condition

const double *startTime* = 0.1;

\_cars.NextT = *startTime*;

for (var i = 0; i < 3; i++)

{

\_cashier1.InAct(new Item(*startTime*));

\_cashier2.InAct(new Item(*startTime*));

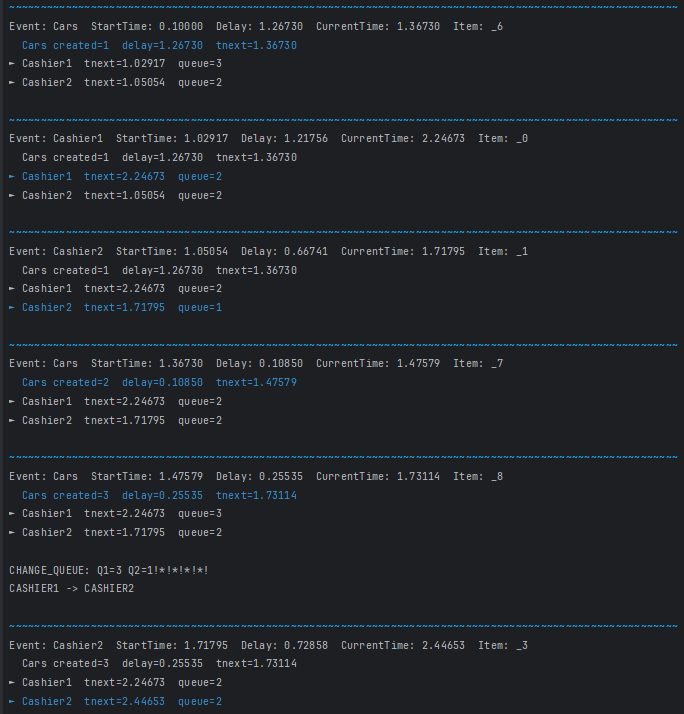
}

Model = new Task2Model(new List<Element> { \_cars, \_cashier1, \_cashier2 });

}

## Вивід процесу

Запустимо симуляцію на час 10, щоб продивитися кроки виконання моделі:

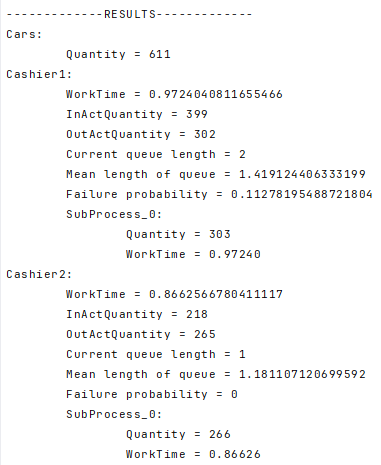


Кроки:

1. Через початкові умови маємо зайнятих касирів та черги по дві машини у кожній. Приїжджає перший клієнт у час 0.1, який стає у чергу до першого касира, оскільки черга до 1-го касира при рівних чергах є пріоритетною.
2. Перший касир опрацював клієнта, перейшов до наступного, черга стала 2.
3. Пункт 2 тільки для другого касира, черга стала 1
4. Приїхав клієнт. Оскільки черга до 2-го касира менша займає її.
5. Приїжджає машина, стає у чергу №1. Черги стають 3 та 2.
6. Касир №2 обробив клієнта, взяв наступного. Черги стали 3 та 1. Останній автомобіль черги №1 переїхав до черги №2. Черги стали 2, 2.

## Результат

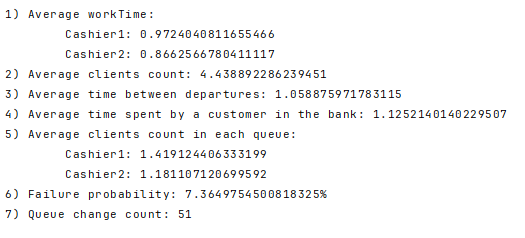
Виконаємо симуляцію на час 300:



Отже:

1. Під’їхало до банку 611 автомобілів, касири загалом обробили 567 та обробляють ще двох. Отже, 42 клієнта не заїхали на обслуговування;
2. Середня черга майже однакова, через переїзди автомобілів, і сягає 2-х;
3. Перший касир працював майже увесь час, і обробив більше ніж другий;
4. В даній системі Failure probability у другого касира завжди буде 0. Оскільки при чергах 3 та 3, наступний клієнт обере 1-го касира. Побачивши, що черга зайнята, він поїде з території банку.

Статистика:



# Завдання 3

## Завдання

Для наступного тексту задачі скласти формалізовану модель масового обслуговування та реалізувати її з використанням побудованого універсального алгоритму (**40 балів**):

У лікарню поступають хворі таких трьох типів:

1) хворі, що пройшли попереднє обстеження і направлені на лікування;

2) хворі, що бажають потрапити в лікарню, але не пройшли повністю попереднє обстеження;

3) хворі, які тільки що поступили на попереднє обстеження.

Чисельні характеристики типів хворих наведені в таблиці:

| Тип хворого | Відносна частота | Середній час реєстрації, хв |
| --- | --- | --- |
| 1 | 0,5 | 15 |
| 2 | 0,1 | 40 |
| 3 | 0,4 | 30 |

При надходженні в приймальне відділення хворий стає в чергу, якщо обидва чергових лікарі зайняті. Лікар, який звільнився, вибирає в першу чергу хворого типу 1.

Після заповнення різноманітних форм у приймальне відділення хворі 1 типу ідуть прямо в палату, а хворі типів 2 і 3 направляються в лабораторію. Троє супровідних розводять хворих по палатах. Хворим не дозволяється направлятися в палату без супровідного. Якщо всі супровідні зайняті, хворі очікують їхнього звільнення в приймальному відділенні. Як тільки хворий доставлений у палату, він вважається таким, що завершив процес прийому до лікарні. Хворі, що спрямовуються в лабораторію, не потребують супроводу. Після прибуття в лабораторію хворі стають у чергу в реєстратуру. Після реєстрації вони ідуть у кімнату очікування, де чекають виклику до одного з двох лаборантів. Після здачі аналізів хворі або повертаються в приймальне відділення (хворий типу 2), або залишають лікарню (хворий типу 1). Після повернення в приймальне відділення хворий, що здав аналізи, розглядається як хворий типу 1.

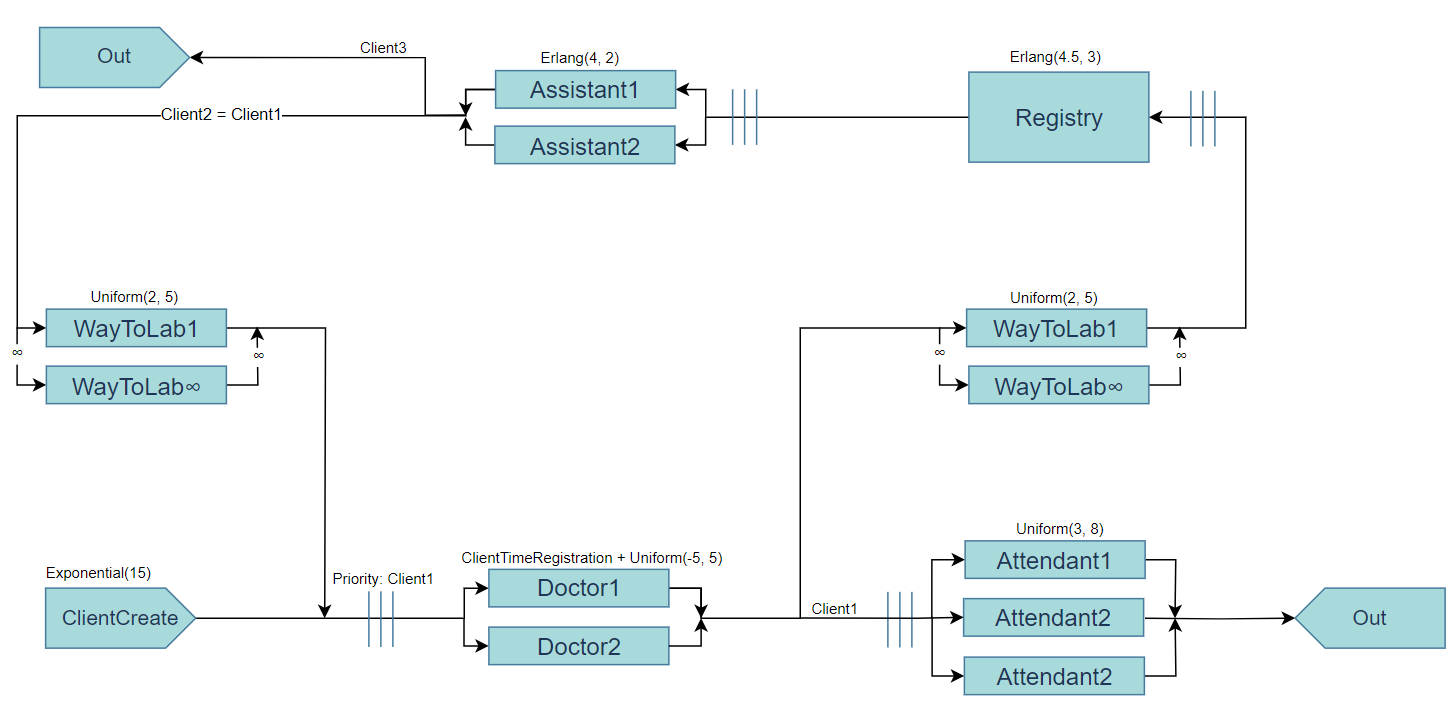
У наступній таблиці приводяться дані по тривалості дій (хв):

| Величина | Розподіл |
| --- | --- |
| Час між прибуттями в приймальне відділення | Експоненціальний з математичним сподіванням 15 |
| Час слідування в палату | Рівномірне від 3 до 8 |
| Час слідування з приймального відділення в лабораторію і навпаки | Рівномірне від 2 до 5 |
| Час обслуговування в реєстратуру лабораторії | Ерланга з математичним сподіванням 4,5 і k=3 |
| Час проведення аналізу в лабораторії | Ерланга з математичним сподіванням 4 і k=2 |

Визначити час, проведений хворим у системі, тобто інтервал часу, починаючи з надходження і закінчуючи доставкою в палату (для хворих типу 1 і 2) або виходом із лабораторії (для хворих типу 3).

Визначити також інтервал між прибуттями хворих у лабораторію.

## Діаграма структури



## Додаткові умови та їх реалізація

Маємо реалізувати:

1. Типи хворих:

public class Client : Item

{

public ClientType ClientType { get; set; }

public double RegistrationTime { get; set; }

public Client(string name, ClientType clientType, double registrationTime, double startTime) : base(startTime)

{

Name = name;

ClientType = clientType;

RegistrationTime = registrationTime;

}

}

public enum ClientType {*Chamber*, *NotExamined*, *Lab*}

1. Створення хворих із різною ймовірністю:

public class CreateClient : Create

{

public CreateClient(Randomizer rand,string name) : base(rand,name) {}

protected override Item CreateItem()

{

var number = new Random().NextDouble();

if (number <= 0.5) return new Client("ChamberClient", ClientType.*Chamber*, 15, CurrT);

if (number <= 0.6)

return new Client("NotExaminedChamberClient",ClientType.*NotExamined*, 40, CurrT);

return new Client("LabClient", ClientType.*Lab*, 30, CurrT);

}

}

1. Пріоритетну чергу до доктора для хворого типу 1:

internal class ClientsQueue : ItemsQueue

{

public ClientsQueue(int limit) : base(limit) {}

public override Item GetItem()

{

var chamberClient = Queue.Find(x => (x as Client).ClientType == ClientType.*Chamber*);

if (chamberClient != null)

{

Queue.Remove(chamberClient);

return chamberClient;

}

return base.GetItem();

}

}

1. Перехід від доктора до наступних елементів в залежності від типу хворого:

public class NextAfterDoctor : NextElementsContainer

{

private Process NextAttendantProcess { get; }

private Process NextWayToLab { get; }

private DoctorProcess \_doctorProcess;

public NextAfterDoctor(DoctorProcess doctorProcess, Process nextAttendantProcess, Process nextWayToLab)

{

\_doctorProcess = doctorProcess;

NextAttendantProcess = nextAttendantProcess;

NextWayToLab = nextWayToLab;

}

protected override Element GetNextElement()

{

var client = \_doctorProcess.Item as Client;

if (client.ClientType == ClientType.*Chamber*)

return NextAttendantProcess;

return NextWayToLab;

}

}

1. Клас DoctorProcess:

public class DoctorProcess : Process

{

public DoctorProcess(int doctorsCount, string name, string subProcessName, int maxQueue = 2147483647)

: base(new UniformRandomizer(-5, 5), doctorsCount, name, maxQueue, subProcessName)=> Queue = new ClientsQueue(maxQueue);

protected override double GetDelay()=> (Item as Client).RegistrationTime + Randomizer.GenerateDelay();

}

1. Генерація розподілу Ерланга:

public class ErlangRandomizer : Randomizer

{

private double TimeMean { get; }

private int k { get; }

public ErlangRandomizer(double timeMean, int k)

{

TimeMean = timeMean;

this.k = k;

}

public override double GenerateDelay() => Enumerable.*Range*(0, k).Select(\_ => -TimeMean \* Math.*Log*(\_random.NextDouble())).Sum();

}

1. Зміна типу клієнта після здачі аналізів:

public class LabAssistanceProcess : Process

{

public LabAssistanceProcess(Randomizer randomizer, int assistanceCount, string name, string subProcessName, int maxQueue = 2147483647) :

base(randomizer, assistanceCount, name, maxQueue, subProcessName) {}

protected override void NextElementsContainerSetup()

{

if (Item is Client { ClientType: ClientType.*NotExamined* } client)

{

client.ClientType = ClientType.*Chamber*;

client.RegistrationTime = 15;

client.Name = "ChamberClient";

}

else NextElementsContainer = null;

}

}

1. Збір статистики:

public class Task3HospitalModel : Model

{

public Task3HospitalModel(List<Element> elements, bool initialStateIsNeeded = false) : base(elements, initialStateIsNeeded) {}

public override void Simulate(double time, double startTime = 0, bool printSteps = false)

{

base.Simulate(time, startTime, printSteps);

Console.*WriteLine*();

Console.*WriteLine*($"1) Average spent time in system: {AverageItemTimeInSystem}");

Console.*WriteLine*($"2) Average time between arrivals in registry: {AverageBetweenInActs\_2()}");

}

private double AverageBetweenInActs\_2()

{

var lab = Elements.OfType<Process>().First(p => p.Name == "Registry");

return lab.TotalTimeBetweenInActs / lab.InActQuantity;

}

}

## Код побудови системи

public class Task3Hospital

{

public Model Model { get; }

public Task3Hospital()

{

CreateClient patients = new CreateClient(new ExponentialRandomizer(15), "Patient");

DoctorProcess doctors = new(2, "Doctors", "Doctor");

Process attendants = new(new UniformRandomizer(3, 8), subProcessCount: 3, name: "Attendants", subProcessName: "Attendant");

Process fromHospitalToLab = new(new UniformRandomizer(2, 5), 25, name: "WayToLab");

Process labRegistry = new(new ErlangRandomizer(4.5, 3), name: "Registry");

LabAssistanceProcess labAssistants = new(new ErlangRandomizer(4, 2), assistanceCount: 2, "Assistants", subProcessName: "Assistant");

Process fromLabToHospital = new(new UniformRandomizer(2, 5), 25, name: "WayToHospital");

patients.NextElementsContainer = new NextElementContainer(doctors);

doctors.NextElementsContainer = new NextAfterDoctor(doctors, attendants, fromHospitalToLab);

fromHospitalToLab.NextElementsContainer = new NextElementContainer(labRegistry);

labRegistry.NextElementsContainer = new NextElementContainer(labAssistants);

labAssistants.NextElementsContainer = new NextElementContainer(fromLabToHospital);

fromLabToHospital.NextElementsContainer = new NextElementContainer(doctors);

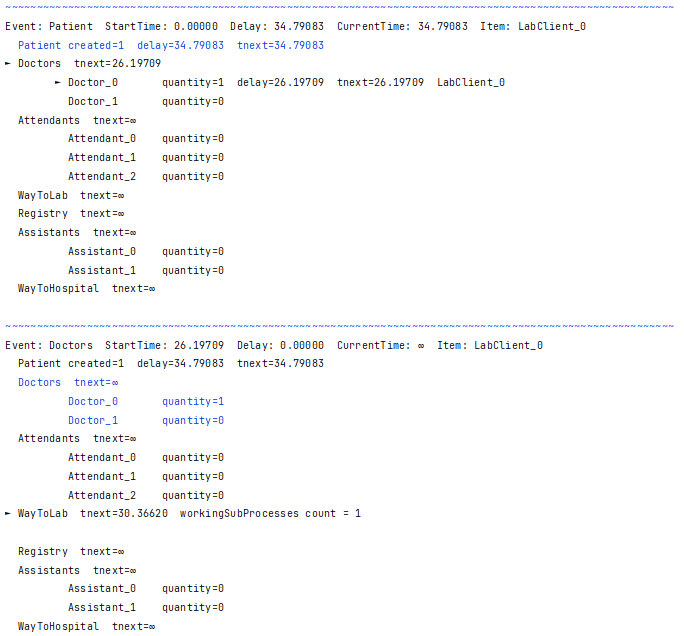
Model = new Task3HospitalModel(new List<Element>() { patients, doctors, attendants, fromHospitalToLab, labRegistry, labAssistants, fromLabToHospital });

}

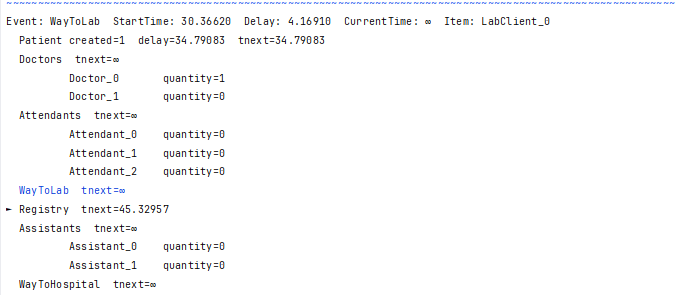
}

**Вивід процесу**

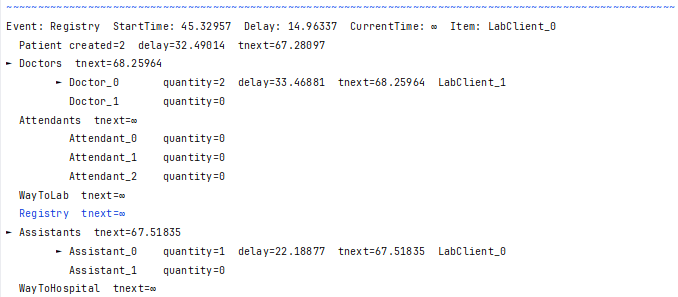
Продемонструємо роботу системи для хворих типу 2 та 3 (тип 2 стає типом 1, таким чином можемо побачити поведінку і для нього). Тип 3:



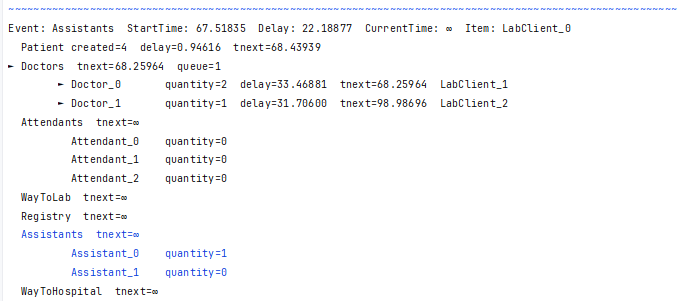
LabClient приходить і потрапляє до доктора №0. Після доктора йде до лабораторії.



Пацієнта реєструють.

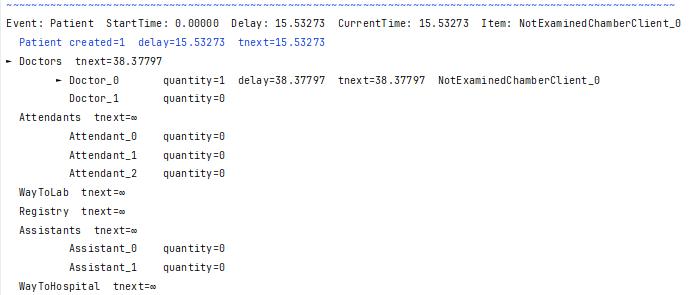


Пацієнта обстежує асистент №0. За цей час до доктора №0 надійшов наступний пацієнт.



Асістент закінчив обстеження LabClient\_0. Хворий покинув систему. За час обстеження доктор №1 прийняв хворого, і утворилася черга розміром 1 у приймальному відділенні.

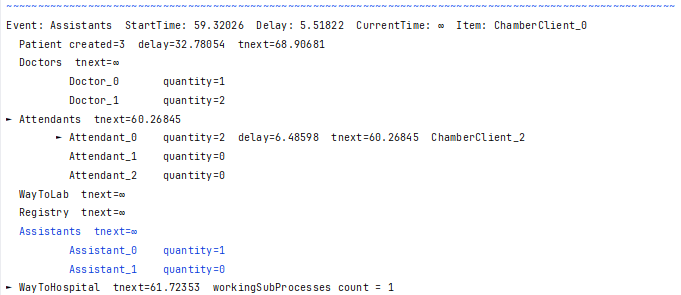
Тепер приклад поведінки пацієнта №2 у системі:



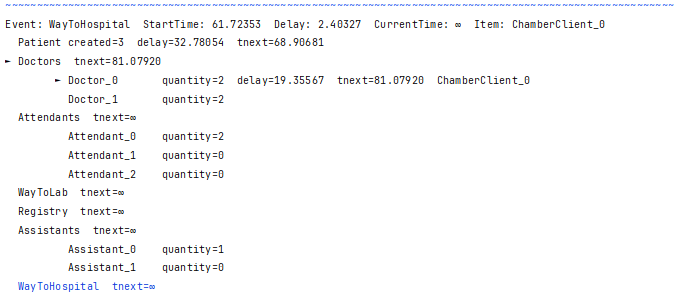
Пацієнт №2 (NotExaminedChamberClient) прийшов у приймальне відділення і його прийняв доктор №0.



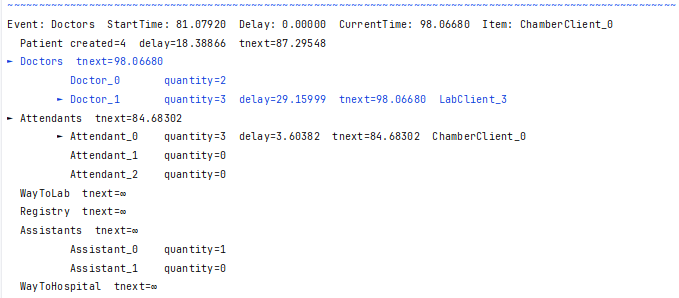
Після доктору наш пацієнт пішов у лабораторію. За час спілкування із лікарем надійшло 2 пацієнти типу 1, 1 з яких був проведений у палату після спілкування із доктором. Пропустимо ідентичні етапи перевірки у лабораторії із хворим типу 3.



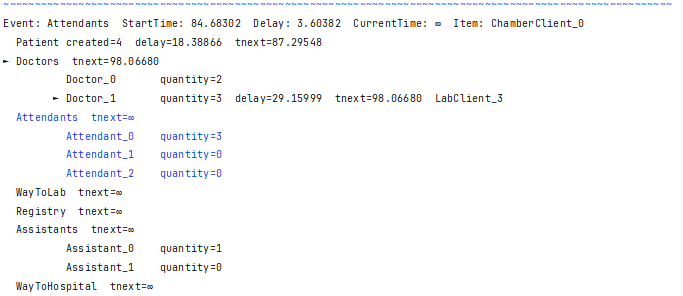
Ассистент закінчив роботу із нашим пацієнтом і він направився до приймального відділення.



Хворий прийшов до лікарні і його прийняв лікар як хворого типу 1.



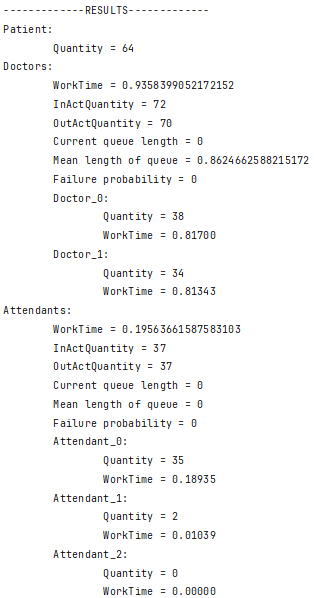
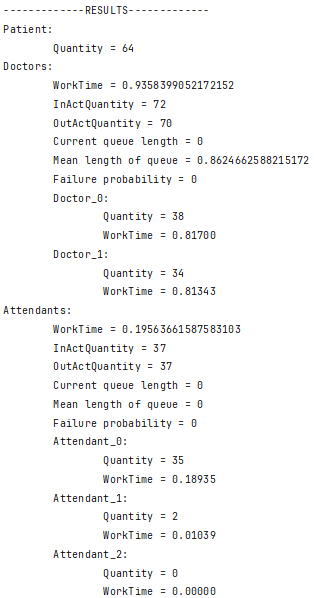
Після спілкування із доктором, хворого забирає супровідний і веде до палати.

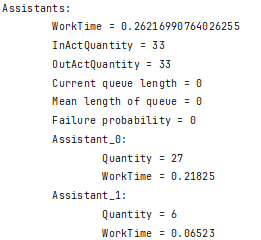
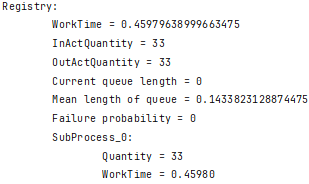


Його провели і він є 3-ім пацієнтом, що надійшов до палати.

## Результат

Проведемо симуляцію на час 1000 хвилин. Статистика працівників:





Статистика переходів:



Додаткова статистика:

1) Average spent time in system: 43.99584379435704

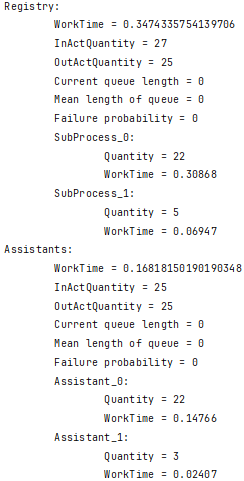
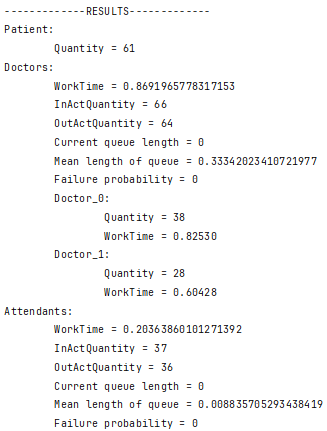
2) Average time between arrivals in registry: 27.322393179796315

Висновки:

1. Доктори працюють майже увесь час і мають чергу в 1 людину в середньому. Оскільки загальна кількість клієнтів, що прийшла до лікарні 64, а доктори виконали 72 сеанси, то можна стверджувати, що 8 хворих прийшли повторно (тип 2).
2. Супровідний №0 провів майже усіх хворих, черги немає. Для економії грошей лікарня може відмовитись від вакансій Супровідний №1 та №2.
3. Реєстратура працює добре, завантаженість 46%, іноді формуються черги до 2 особ. Для зниження часу хворого у системі можна додати додаткового працівника у реєстратуру.
4. Ассистенти працюють добре, черг немає. Формується ситуація, що ассистент №0 не справиться сам. Але сильного навантаження на №1 немає. На місце ассистента №1 можна брати інтернів.
5. З переходів видно, що на частку тих, хто прийшов до лабораторії, 75% просто здають аналізи. Можливо реалізувати пріоритет у черзі для хворих, що будуть лягати у лікарню на етапі реєстрації. Це має зменшити час хворого у системі. Або виконати пункт 3.

## Результати нової моделі

Спробуємо запустити нову модель без супровідних №1 та №2, та додамо нового працівника у реєстатуру:



1) Average spent time in system: 32.53513388382148

2) Average time between arrivals in registry: 35.49048691635963

Як видно з результатів, середній час пацієнта у системи знизився з 44 хвилин до 33. Супровідник справляється сам. Завантаженість працівників реєстратури подібна до ассистентської. Різниця полягає лише у часі процесів. В реєстратурі час роботи завеликий. Отже, по можливості, для налагодження системи необхідно скоротити час самої реєстрації. Це призведе до більш швидкої здачі аналізів, а це може призвести до збільшення черги на прийом до лікаря. В такому разі необхідно буде додати нову вакансію на посаду лікар №3.