

Міністерство освіти і науки України

Національний технічний університет України

„КПІ імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки Кафедра інформатики і програмної інженерії

**Звіт**

Комп’ютерний практикум №3  
«ПОБУДОВА ІМІТАЦІЙНОЇ МОДЕЛІ СИСТЕМИ З ВИКОРИСТАННЯМ ФОРМАЛІЗМУ МОДЕЛІ МАСОВОГО ОБСЛУГОВУВАННЯ»

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*Замрій Олексій ІП-96*

**Перевірив**

(прізвище, ім'я, по батькові)

Київ 2022

Зміст

[1 Завдання 3](#_Toc118735675)

[2 Виконання 4](#_Toc118735676)

[2.1 Програмна реалізація алгоритму 4](#_Toc118735677)

[2.2 Хід роботи 16](#_Toc118735678)

[2.2.1 Завдання 1-2 16](#_Toc118735679)

[2.2.2 Завдання 3-4 17](#_Toc118735680)

[Висновок 18](#_Toc118735681)

# Завдання

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

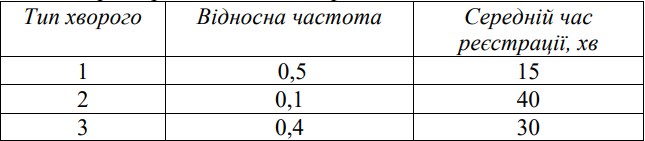
2. Для наступного тексту задачі скласти формалізовану модель масового обслуговування та реалізувати її з використанням побудованого універсального алгоритму (30 балів):

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 такі 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.

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

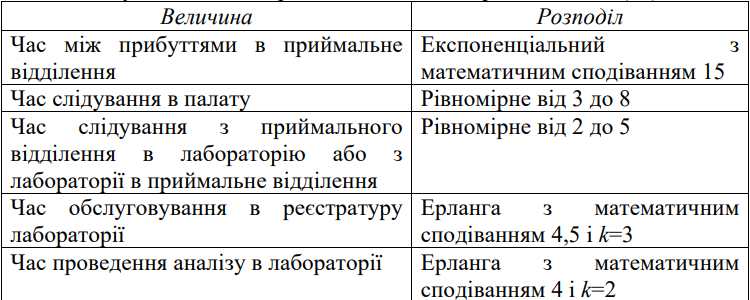
У лікарню поступають хворі таких трьох типів: 1) хворі, що пройшли попереднє обстеження і направлені на лікування; 2) хворі, що бажають потрапити в лікарню, але не пройшли повністю попереднє обстеження; 3) хворі, які тільки що поступили на попереднє обстеження. Чисельні характеристики типів хворих наведені 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 тих хворих, що вже пройшли попереднє обстеження. 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 (якщо їх приймають у лікарню), або залишають лікарню (якщо їм було призначено тільки попереднє обстеження). 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 або з лабораторії в приймальне відділення 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.

# Виконання

## Програмна реалізація алгоритму

Лістинг файлу **Program.cs**

using System;

using System.Collections.Generic;

namespace Lab2

{

class Program

{

static void Main(string[] args)

{

Model model = new Model();

Create creator = new Create("Creator", 0.5);

creator.NextTime = 0.1;

model.SetNextElement(creator);

Process cashbox1 = new Process("Cashbox#1", 1, 0.3, "norm", queueLimit: 3);

cashbox1.SetCondition(StepBackCondition);

cashbox1.Access = false;

cashbox1.Queue = 2;

Process cashbox2 = new Process("Cashbox#2", 1, 0.3, "norm", queueLimit: 3);

cashbox2.SetCondition(StepBackCondition);

cashbox2.Access = false;

cashbox2.Queue = 2;

ProcessBlock bank = new ProcessBlock("Bank", 0, workers: new Process[] { cashbox1, cashbox2 });

bank.SetChooseCondition(SelectionCondition);

model.SetNextElement(bank);

Counter final = new Counter("Final counter");

model.SetNextElement(final);

model.Simulate(100);

bool StepBackCondition(Component c)

{

List<Component> receivers = c.Sender.Receivers;

Process neigbor = (Process)receivers[(receivers.IndexOf(c) + 1) % receivers.Count];

return c.Queue - neigbor.Queue >= 2;

}

Component SelectionCondition(List<Component> components)

{

if (components[0].FirstLevelAccess)

return components[0];

if (components[1].FirstLevelAccess)

return components[1];

if (components[0].Queue <= components[1].Queue)

return components[0];

return components[1];

}

}

}

}

Лістинг файлу Model.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab2

{

class Model

{

private List<Component> model;

private double currTime, nextTime;

public Model()

{

model = new List<Component>();

nextTime = 0.0;

currTime = 0.0;

}

public void SetNextElement(Component c)

{

if (model.Count != 0)

model[model.Count - 1].SetNext(c);

model.Add(c);

}

public void Simulate(double duration)

{

while(currTime < duration)

{

Component currComponent = null;

nextTime = Double.MaxValue;

foreach(Component c in model)

{

if(c.GetNextTime() < nextTime)

{

nextTime = c.GetNextTime();

currComponent = c.GetNextComponent();

}

}

Console.WriteLine($"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_It's {nextTime} on the clock and it's time for {currComponent.name}\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

foreach (Component c in model)

c.GetStatistics(nextTime - currTime);

currTime = nextTime;

foreach (Component c in model)

c.SetCurrTime(currTime);

foreach (Component c in model)

if (c.GetNextTime() == currTime)

c.OutAct();

foreach (Component c in model)

{

c.GetState();

Console.WriteLine("||\n\\/");

}

Console.WriteLine("End.\n\n");

}

GetResult();

}

private void GetResult()

{

foreach (Component c in model) c.SetCurrTime(currTime);

Console.WriteLine(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>RESULTS<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<");

foreach(Component c in model)

{

c.GetResult();

Console.WriteLine("||\n\\/");

}

Console.WriteLine("End.\n");

}

}

}

Лістинг файлу Component.cs

using System;

using System.Collections.Generic;

namespace Lab2

{

public class Task

{

public string name;

public double startTime = 0;

public double frequency, addFrequency;

public double processingDelay, addProcessingDelay;

Stack<Component> path;

public Component Path

{

get => path.Peek();

set => path.Push(value);

}

public Task(string name = "#NoName Task", double frequency = -1, double addFrequency = -1, double processcingDelay = -1, double addProcessingDelay = -1)

{

path = new Stack<Component>();

this.name = name;

this.frequency = frequency;

this.addFrequency = addFrequency;

this.processingDelay = processcingDelay;

this.addProcessingDelay = addProcessingDelay;

}

public Task Copy()

{

return new Task(this.name, this.frequency, this.processingDelay, this.addProcessingDelay);

}

public Component KickBack() => path.Pop();

}

public abstract class Component

{

protected static Random rand = new Random();

protected List<Component> nextComponent;

protected double currTime, nextTime, avgQueue;

protected double basicDelay, additionalDelay;

protected QueuePickCondition queuePickCondition;

protected ChooseCondition chooseCondition;

protected Action addStatisicAction;

protected int itemsCount, dropsOut;

public string name, distribution;

protected List<int> chances;

protected List<Task> tasks;

protected bool selection;

protected List<double> buffer = new List<double>();

public double NextTime

{

get => nextTime;

set => nextTime = value;

}

public virtual bool FirstLevelAccess

{

get => true;

}

public virtual int SecondLevelAccess

{

get => Int16.MaxValue;

}

public virtual int Queue

{

get => 0;

set { }

}

public Component Sender

{

get => tasks[tasks.Count - 1].Path;

}

public List<Component> Receivers

{

get => nextComponent;

}

public bool SelectionFlag

{

get => selection;

set => selection = value;

}

public Component(string name = "NoName", double delay = 1.0, double addDelay = 0.5, string d = "exp", bool selection = true)

{

nextComponent = new List<Component>();

chances = new List<int>();

tasks = new List<Task>();

chooseCondition = DefaultSelectionCondition;

queuePickCondition = DefaultQueuePickCondition;

addStatisicAction = null;

additionalDelay = addDelay;

this.selection = selection;

basicDelay = delay;

this.name = name;

distribution = d;

currTime = 0.0;

nextTime = 0.0;

dropsOut = 0;

}

/// <summary>

/// Integers representation of chances ratio.

/// If there no params - all receivers gets equal chance ration.

/// If there not enough params - rest components have 1 ratio chance.

/// If there more params than needs - rest params will be ignored.

/// </summary>

public void SetChoosingChances(params int[] chances)

{

this.chances.Clear();

for (int i = 0; i < nextComponent.Count; i++)

this.chances.Add(1);

int index = 0;

while(index < chances.Length && index < this.chances.Count)

{

this.chances[index] = chances[index];

index++;

}

}

protected Component ChoosingByChances()

{

List<Component> enableComponents = new List<Component>();

List<int> enableChances = new List<int>();

for(int i = 0; i < nextComponent.Count; i++)

{

if(nextComponent[i].FirstLevelAccess || nextComponent[i].SecondLevelAccess > 0)

{

enableComponents.Add(nextComponent[i]);

enableChances.Add(chances[i]);

}

}

int r = rand.Next(Sum(enableChances));

for(int i = 0; i < enableChances.Count; i++)

{

r -= enableChances[i];

if (r < 0)

return enableComponents[i];

}

return null;

}

public void SetChooseCondition(ChooseCondition cond) => chooseCondition = cond;

public void SetQueuePickCondition(QueuePickCondition cond) => queuePickCondition = cond;

public void SetAdditionalStatisticAction(Action action) => addStatisicAction = action;

protected double GetDelay(double delay, double addDelay)

{

delay = (delay == -1) ? this.basicDelay : delay;

addDelay = (addDelay == -1) ? this.additionalDelay : addDelay;

switch (distribution)

{

case "exp":

return Generator.Exp(delay);

case "unif":

return Generator.Unif(delay, addDelay);

case "norm":

return Generator.Norm(delay, addDelay);

case "erl":

return Generator.Norm(delay, (int)addDelay);

default:

return Generator.Exp(delay);

}

return basicDelay;

}

public virtual double GetNextTime() => nextTime;

public virtual void OutAct()

{

itemsCount++;

}

public abstract int InAct(Task task);

internal virtual void SetCurrTime(double time) => currTime = time;

internal void SetDistributionType(string d) => distribution = d;

public virtual void GetStatistics(double d) { }

public abstract void GetState();

public abstract void GetResult();

public virtual void SetNext(params Component[] components)

{

nextComponent.Clear();

foreach(Component c in components)

nextComponent.Add(c);

SetChoosingChances();

}

public virtual void AddNext(Component n)

{

nextComponent.Add(n);

chances.Add(Avg(chances));

}

public virtual void ClearNext()

{

nextComponent.Clear();

chances.Clear();

}

public virtual Component GetNextComponent() => this;

public virtual void CheckStepsBack() { }

public void KickBack()

{

Task last = tasks[tasks.Count - 1];

tasks.RemoveAt(tasks.Count - 1);

last.KickBack().InAct(last);

}

protected Component DefaultSelectionCondition(List<Component> components)

{

if (components is null || components.Count == 0)

{

Console.WriteLine("Something wrong: Receiver List is null or empty.");

return null;

}

foreach (Process p in nextComponent)

{

if (p.FirstLevelAccess)

return p;

}

Process selected = null;

int max = 0;

foreach (Process p in nextComponent)

{

if (p.SecondLevelAccess > max)

{

selected = p;

max = p.SecondLevelAccess;

}

}

if (selected is null)

return null;

return selected;

}

protected int DefaultQueuePickCondition(List<Task> tasks) => 1;

protected static int Avg(List<int> chances) => Sum(chances) / chances.Count;

protected static int Sum(List<int> chances)

{

int sum = 0;

foreach (int i in chances)

sum += i;

return sum;

}

}

class Create : Component

{

List<Task> basicTasks;

List<int> taskChances;

public Create(string name, double delay, Component receiver = null, Task example = null):base(name, delay)

{

basicTasks.Add((example is null) ? new Task(name + " product", delay) : example);

if (!(receiver is null)) nextComponent.Add(receiver);

SetCreatingChances();

SetChoosingChances();

}

public void SetTask(Task example)

{

basicTasks.Add(example);

taskChances.Add(1);

}

public void SetCreatingChances(params int[] chances)

{

this.taskChances.Clear();

for (int i = 0; i < basicTasks.Count; i++)

this.taskChances.Add(1);

int index = 0;

while (index < chances.Length && index < this.taskChances.Count)

{

this.taskChances[index] = chances[index];

index++;

}

}

public override int InAct(Task task) => -1;

public override void OutAct()

{

base.OutAct();

Task product = GetProduct();

product.startTime = currTime;

product.Path = this;

if (nextComponent.Count > 0)

{

nextTime = currTime + base.GetDelay(product.frequency, product.addFrequency);

if (nextComponent.Count == 1)

nextComponent[0].InAct(product);

else

{

Component toWork = (selection) ? chooseCondition(nextComponent) : ChoosingByChances();

if (toWork is null)

dropsOut++;

else

toWork.InAct(product);

}

}

else Console.WriteLine("Something wrong: Create block has no receiver.");

}

private Task GetProduct()

{

int r = rand.Next(Sum(taskChances));

for (int i = 0; i < taskChances.Count; i++)

{

r -= taskChances[i];

if (r < 0)

return basicTasks[i];

}

return null;

}

public override void GetState() => Console.WriteLine($"Creation block: {name} > item passed = {itemsCount} | next action at: {nextTime}s;");

public override void GetResult() => Console.WriteLine($"Creation block: {name} > item passed = {itemsCount};");

}

class Process: Component

{

protected int queue, queueLimit;

private bool isAccessible, innerQueue;

private KickBackCondition? cond;

public override int Queue

{

get => queue;

set

{

queue = value;

List<Task> newTasks = new List<Task>();

newTasks.Add(tasks[0]);

for (int i = 0; i < value; i++)

newTasks.Add(new Task());

tasks = newTasks;

}

}

public bool Access

{

set

{

isAccessible = value;

if (!value)

{

tasks.Add(new Task());

nextTime = currTime + GetDelay(tasks[0].processingDelay, tasks[0].addProcessingDelay);

}

else

tasks.RemoveAt(0);

}

}

public override bool FirstLevelAccess

{

get => isAccessible;

}

public override int SecondLevelAccess

{

get => innerQueue ? queueLimit - queue : 0;

}

public Process(string name, double delay = 1.0, double addDelay = 0.5, string distribution = "exp", int queueLimit = Int32.MaxValue, Component receiver = null, bool innerQueue = true) : base(name, delay, addDelay, distribution)

{

nextTime = Double.MaxValue;

queue = 0;

this.queueLimit = queueLimit;

this.innerQueue = innerQueue;

if (queueLimit <= 0) this.innerQueue = false;

isAccessible = true;

if (!(receiver is null)) nextComponent.Add(receiver);

SetChoosingChances();

cond = null;

buffer.Add(0);

buffer.Add(0);

}

public void SetCondition(KickBackCondition c) => cond = c;

public override int InAct(Task task)

{

if(isAccessible)

{

isAccessible = false;

tasks.Add(task);

nextTime = currTime + GetDelay(task.processingDelay, task.addProcessingDelay);

}

else

{

if (innerQueue && queue < queueLimit)

{

tasks.Add(task);

queue++;

}

else

{

dropsOut++;

return -1;

}

}

return 1;

}

public override void OutAct()

{

if(!isAccessible) base.OutAct();

Task forSending = tasks[0];

forSending.Path = this;

if (queue != 0)

{

queue--;

isAccessible = false;

int chosenTaskIndex = queuePickCondition(tasks);

tasks[0] = tasks[chosenTaskIndex];

tasks.RemoveAt(chosenTaskIndex);

nextTime = currTime + base.GetDelay(tasks[0].processingDelay, tasks[0].addProcessingDelay);

}

else

{

tasks.RemoveAt(0);

nextTime = Double.MaxValue;

isAccessible = true;

}

if (nextComponent.Count > 0)

{

if (nextComponent.Count == 1)

nextComponent[0].InAct(forSending);

else

if (selection)

chooseCondition(nextComponent).InAct(forSending);

else

ChoosingByChances().InAct(forSending);

}

}

public override void CheckStepsBack()

{

if (!(cond is null))

{

while (queue > 0 && cond(this))

{

queue--;

buffer[1]++;

KickBack();

}

}

}

public override void GetStatistics(double d)

{

avgQueue += queue \* d;

if (isAccessible)

buffer[0] += queue \* d;

else

buffer[0] += (queue + 1) \* d;

}

public override void GetState() =>

Console.WriteLine("Process: {0} > next action at: {1}s | isAccesible: {2} | queue: {3} | kickBacks: {4};", name, (nextTime == Double.MaxValue) ? -1 : nextTime, isAccessible, queue, buffer[1]);

public override void GetResult()

{

Console.WriteLine($"Process: {name} > item passed: {itemsCount} | drops out: {dropsOut};");

Console.WriteLine($"Average queue: {avgQueue / currTime} | items losing probability: {dropsOut / ((double)itemsCount + dropsOut)}");

Console.WriteLine($"Average load: {buffer[0] / currTime} | Total kickBacks: {buffer[1]}");

}

}

class ProcessBlock: Process

{

private Component nearestComponent;

public override bool FirstLevelAccess

{

get {

foreach (Process p in nextComponent) if(p.FirstLevelAccess) return true;

return false;

}

}

public override int SecondLevelAccess

{

get => queueLimit - queue;

}

public ProcessBlock(string name, int queueLimit = Int32.MaxValue, Component receiver = null, params Process[] workers) : base(name, 1.0, 0.5, "exp", queueLimit, receiver)

{

if (!(workers is null))

{

foreach (Process p in workers)

nextComponent.Add(p);

}

SetChoosingChances();

}

internal override void SetCurrTime(double time)

{

foreach (Process p in nextComponent)

p.SetCurrTime(time);

currTime = time;

}

public override int InAct(Task task)

{

Component toWork = (selection) ? chooseCondition(nextComponent) : ChoosingByChances();

if (toWork is null)

{

if (queue < queueLimit)

{

tasks.Add(task);

queue++;

}

else

{

dropsOut++;

return -1;

}

}

else

{

task.Path = this;

toWork.InAct(task);

}

return 1;

}

public override void OutAct()

{

foreach (Component p in nextComponent)

if (p.GetNextTime() == nextTime)

{

p.OutAct();

itemsCount++;

if (queue != 0)

{

int chosenTaskIndex = queuePickCondition(tasks);

tasks[chosenTaskIndex].Path = this;

p.InAct(tasks[chosenTaskIndex]);

tasks.RemoveAt(chosenTaskIndex);

queue--;

}

}

foreach (Component c in nextComponent)

c.CheckStepsBack();

}

public override double GetNextTime()

{

nextTime = Double.MaxValue;

foreach (Component p in nextComponent)

{

if (p.GetNextTime() <= nextTime)

{

nextTime = p.GetNextTime();

nearestComponent = p.GetNextComponent();

}

}

return nextTime;

}

public override Component GetNextComponent() => nearestComponent;

public override void GetStatistics(double d)

{

int sum = 0;

foreach(Process p in nextComponent)

{

p.GetStatistics(d);

}

avgQueue += queue \* d;

}

public override void GetState()

{

Console.WriteLine("===========================================================================================");

for (int i = 0; i < nextComponent.Count - 1; i++)

{

nextComponent[i].GetState();

Console.WriteLine("||");

}

nextComponent[nextComponent.Count - 1].GetState();

Console.WriteLine("-------------------------------------------------------------------------------------------");

Console.WriteLine("\nProcess block: {0} > next action at: {1}s | queue: {2};", name, (nextTime == Double.MaxValue) ? -1 : nextTime, queue);

Console.WriteLine("===========================================================================================");

}

public override void GetResult()

{

Console.WriteLine("===========================================================================================");

for (int i = 0; i < nextComponent.Count - 1; i++)

{

nextComponent[i].GetResult();

Console.WriteLine("||");

}

nextComponent[nextComponent.Count - 1].GetResult();

Console.WriteLine("-------------------------------------------------------------------------------------------");

Console.WriteLine($"\nProcess block: {name} > item passed: {itemsCount} | drops out: {dropsOut};");

Console.WriteLine($"Average queue: {itemsCount / currTime} | items losing probability: {dropsOut / ((double)itemsCount + dropsOut)}");

Console.WriteLine("===========================================================================================");

}

public override void SetNext(params Component[] components)

{

foreach (Process p in nextComponent)

p.SetNext(components);

}

public override void AddNext(Component component)

{

foreach (Process p in nextComponent)

p.AddNext(component);

}

public override void ClearNext()

{

foreach (Process p in nextComponent)

p.ClearNext();

}

}

class Counter : Component

{

List<double> nextTimes;

private bool delayBlock;

public bool DelayBlock { get => delayBlock; set => delayBlock = value; }

public Counter(string name, double delay = 1.0, double addDelay = 0.5, string distribution = "exp") : base(name, delay, addDelay, distribution)

{

delayBlock = false;

buffer.Add(0);

buffer.Add(-1);

buffer.Add(0);

}

public override int InAct(Task task)

{

tasks.Add(task);

if (delayBlock)

{

nextTimes.Add(currTime + GetDelay(task.processingDelay, task.addProcessingDelay));

FindNext();

}

itemsCount++;

buffer[0] += currTime - task.startTime;

if (buffer[1] == -1)

buffer[1] = currTime;

else

{

buffer[2] += currTime - buffer[1];

buffer[1] = currTime;

}

if(!delayBlock)

OutAct();

return 1;

}

public override void OutAct()

{

if (nextComponent.Count > 0)

{

if (nextComponent.Count == 1)

nextComponent[0].InAct(tasks[(int)nextTime]);

else

if (selection)

chooseCondition(nextComponent).InAct(tasks[(int)nextTime]);

else

ChoosingByChances().InAct(tasks[(int)nextTime]);

}

if (delayBlock)

{

tasks.RemoveAt((int)nextTime);

nextTimes.RemoveAt((int)nextTime);

nextTime = -1;

FindNext();

}

}

public override void GetState() => Console.WriteLine($"Counter: {name} > Item passed: {itemsCount};");

public override void GetResult()

{

Console.WriteLine($"Counter: {name} > Item passed: {itemsCount} | AvgTime per unit: {buffer[0] / itemsCount}s");

Console.WriteLine($"AvgInterval between units: {buffer[2] / itemsCount}");

}

public override double GetNextTime()

{

if(!delayBlock)

return Double.MaxValue;

return nextTimes[(int)nextTime];

}

private void FindNext()

{

if (nextTime == -1 && nextTimes.Count > 0)

nextTime = 0;

for (int i = 0; i < nextTimes.Count; i++)

{

if (nextTimes[i] < nextTimes[(int)nextTime])

nextTime = i;

}

}

}

public delegate Component ChooseCondition(List<Component> components);

public delegate bool KickBackCondition(Component component);

public delegate int QueuePickCondition(List<Task> tasks);

}

Лістинг файлу **Generator.cs**

using System;

using System.Collections.Generic;

using System.Text;

using MathNet.Numerics;

namespace Lab2

{

static class Generator

{

static Random rand = new Random();

public static double Exp(double l)

{

double r = rand.NextDouble();

while(r == 0.0) r = rand.NextDouble();

r = -l \* Math.Log(r);

return r;

}

public static double Unif(double a, double b)

{

double r = rand.NextDouble();

while (r == 0.0) r = rand.NextDouble();

r = a + r \* (b - a);

return r;

}

public static double Norm(double a, double s)

{

return a + s \* StrangeSum();

}

public static double Erlang(double B, uint m)

{

double r = rand.NextDouble();

return (Math.Pow(r, m - 1) \* Math.Exp(r / B)) / (Math.Pow(B, m) \* SpecialFunctions.Gamma(m));

}

private static double StrangeSum()

{

double sum = 0;

for(int i = 0; i < 12; i++)

{

double r = rand.NextDouble();

while (r == 0.0) r = rand.NextDouble();

sum += r;

}

return sum - 6;

}

}

}

## Хід роботи

### Завдання 1

Для виконання першого завдання, алгоритм з другої лабораторної роботи будо модифіковано додаванням можливості переходу з компонентів то декількох наступних компонентів моделі. При цьому є можливість задати спосіб переходу:

* Перехід з вірогідністю, яку можна задавати цілими числами – як пропорцію відношень (за замовчуванням усі такі числа = 1, вірогідність у всіх рівна)
* Перехід за пріоритетом; функція пріоритету може визначатись ззовні (за замовчуванням в пріоритеті вільний для виконання або компонент з найбільшим вільним місцем у черзі).

### Завдання 2

Описана задача у другому завданні виражається такою схемою:

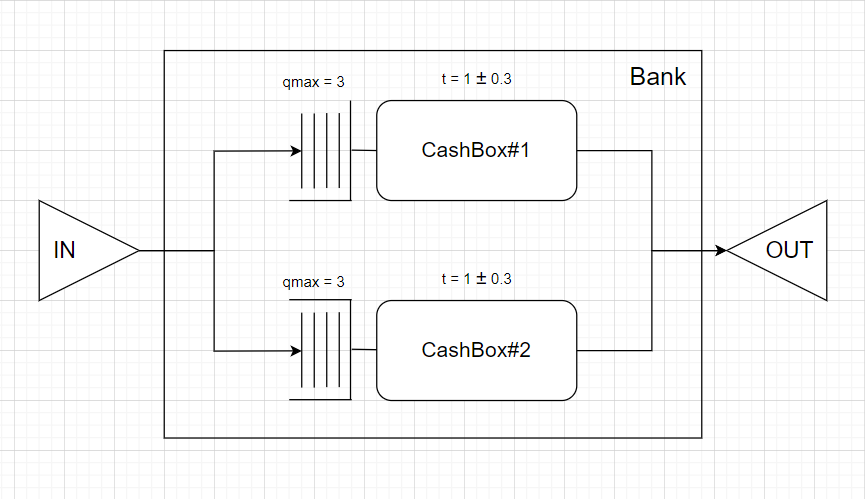


Рисунок 2.2 – Схематичне зображення поставленої задачі

Модель описана у файлі **Program.cs**:

using System;

using System.Collections.Generic;

namespace Lab2

{

class Program

{

static void Main(string[] args)

{

Model model = new Model();

Create creator = new Create("Creator", 0.5);

creator.NextTime = 0.1;

model.SetNextElement(creator);

Process cashbox1 = new Process("Cashbox#1", 1, 0.3, "norm", queueLimit: 3);

cashbox1.SetCondition(StepBackCondition);

cashbox1.Access = false;

cashbox1.Queue = 2;

Process cashbox2 = new Process("Cashbox#2", 1, 0.3, "norm", queueLimit: 3);

cashbox2.SetCondition(StepBackCondition);

cashbox2.Access = false;

cashbox2.Queue = 2;

ProcessBlock bank = new ProcessBlock("Bank", 0, workers: new Process[] { cashbox1, cashbox2 });

bank.SetChooseCondition(SelectionCondition);

model.SetNextElement(bank);

Counter final = new Counter("Final counter");

model.SetNextElement(final);

model.Simulate(100);

bool StepBackCondition(Component c)

{

List<Component> receivers = c.Sender.Receivers;

Process neigbor = (Process)receivers[(receivers.IndexOf(c) + 1) % receivers.Count];

return c.Queue - neigbor.Queue >= 2;

}

Component SelectionCondition(List<Component> components)

{

if (components[0].FirstLevelAccess)

return components[0];

if (components[1].FirstLevelAccess)

return components[1];

if (components[0].Queue <= components[1].Queue)

return components[0];

return components[1];

}

}

}

}

Запустимо модель на симуляцію.

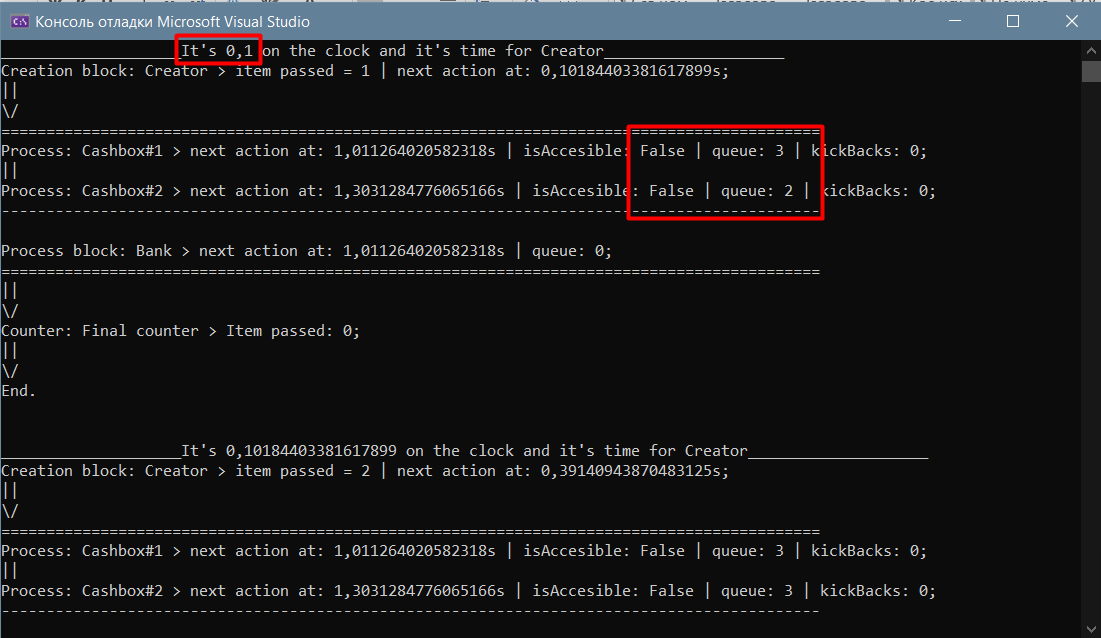


Рисунок 2.2 – Результат запуску алгоритму за схемою (початкові дані)

Можна побачити, що на старті обидві каси вже зайняті, а черги біля них займають чотири машини – по дві на касу. Перша дія – генерація відвідувача у момент часу 0.1, який як сказано у завданні обирає ліву касу при рівних чергах.

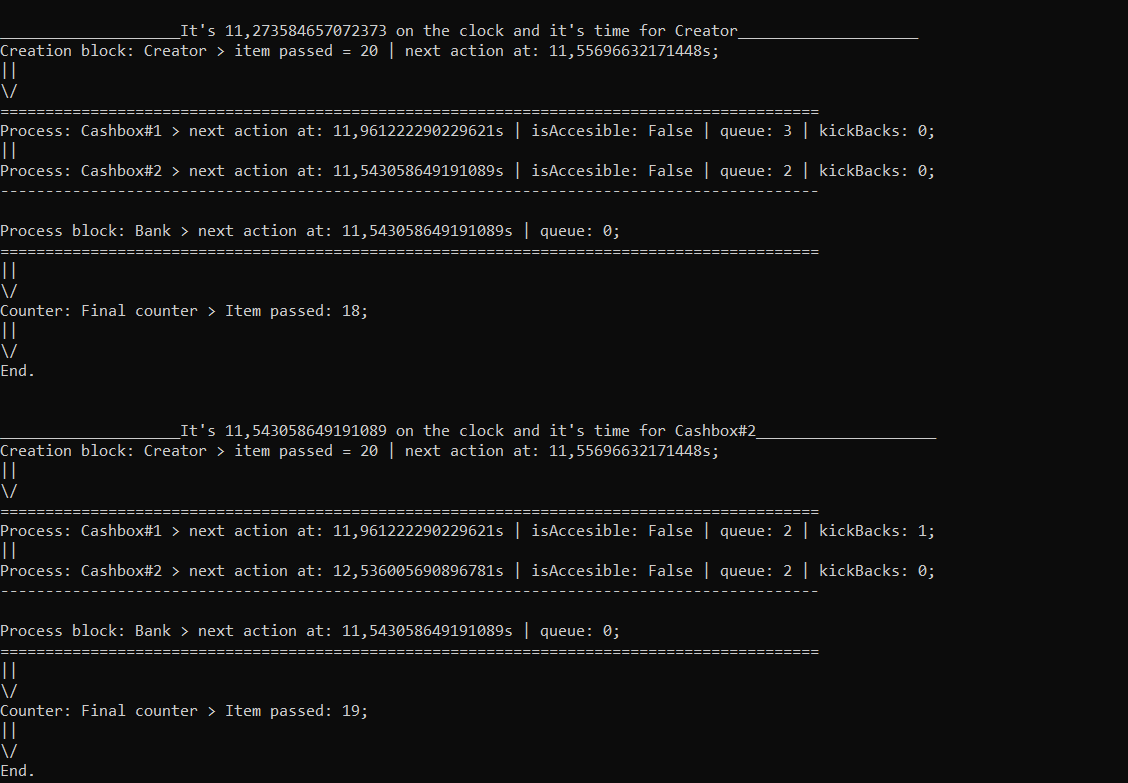


Рисунок 2.3 – Результат запуску алгоритму за схемою (приклад перехід відвідувача у іншу чергу)

Як можна побачити: після дії створення відвідувача, у першій черзі 3 відвідувача, у другій – 2. Наступна дія – закінчення процесу другої каси, таким чином в першій черзі буде 3 відвідувача, а у другій – 1. Тому з першої черги відвідувач переходить до другої.

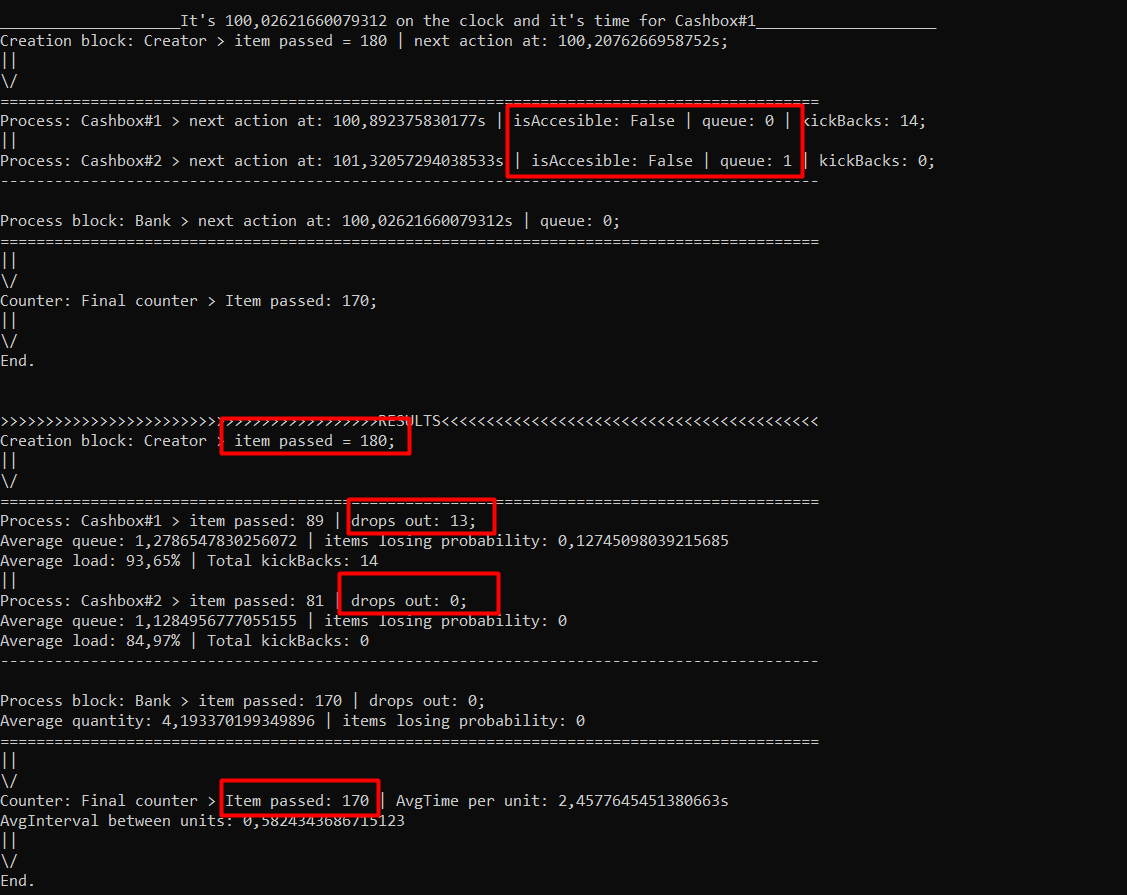


Рисунок 2.4 – Результат запуску алгоритму за схемою (результати)

Для перевірки цілісності, порахуємо кількість вхідних та вихідних вимог:

* Згенеровано було 180 + 6 (4 у чергах та 2 у процесах)
* 186 – 13 (не вмістилось) – 3 (залишились у чергах та процесах)
* = 170 – скільки і у результаті вийшло.

Тож, щодо результатів:

1. середнє завантаження: 1 касир – 93.65%, 2 касир – 84.97%;
2. середня кількість клієнтів у банку: 4.2 клієнти на одиницю часу;
3. середній інтервал від’їзду клієнтів після обслуговування: 0.58 часу;
4. середній час перебування клієнту у банку: 2.45 часу;
5. середнє число клієнтів у чергах: 1 черга – 1.278, 2 черга – 1.128 клієнтів на одиницю часу.
6. відсоток вибувших клієнтів: 12.74%
7. число змін черги клієнтами: 13

### Завдання 3

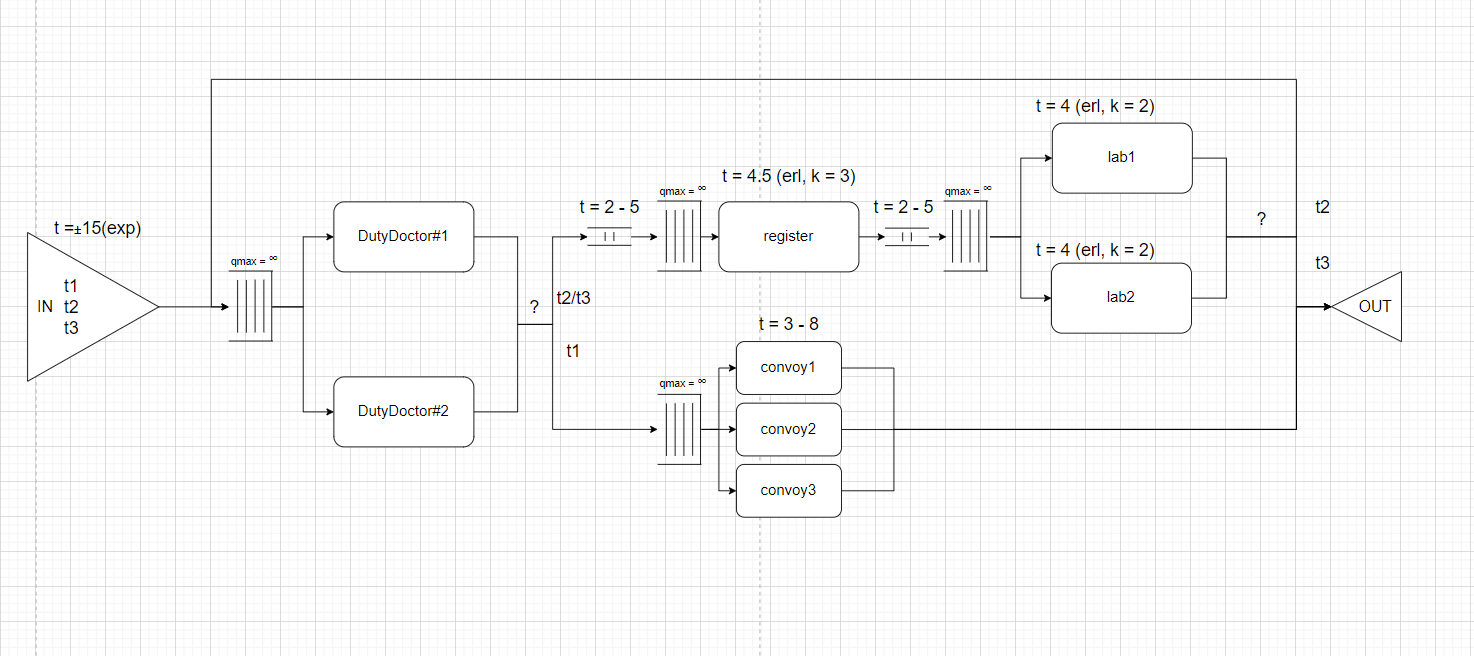
Завдання 3 можна описати такою схемою:

Рисунок 2.5 – Схематичне зображення поставленої задачі

Для реалізації такої схеми було необхідно модифікувати алгоритм, додавши опис вимог, що курсують системою:

public class Task

{

public string name;

public double startTime = 0;

public double processingDelay, addProcessingDelay;

Stack<Component> path;

public Component Path

{

get => path.Peek();

set => path.Push(value);

}

public Task(string name = "#NoName Task", double processingDelay = -1, double addProcessingDelay = -1)

{

path = new Stack<Component>();

this.name = name;

this.processingDelay = processingDelay;

this.addProcessingDelay = addProcessingDelay;

}

public Task Copy()

{

return new Task(this.name, this.processingDelay, this.addProcessingDelay);

}

public Component KickBack() => path.Pop();

}

Цей клас зберігає у собі особливості певної вимогу: час створення, особливості затримки у процесах, пройдений по системі шлях.

Конструювання схеми у програмі у файлі Program.cs:

using System;

using System.Collections.Generic;

namespace Lab2

{

class Program

{

static void Main(string[] args)

{

Model model = new Model();

Task cure1 = new Task("Sick#1", 15);

Task cure2 = new Task("Sick#2", 40);

Task cure3 = new Task("Sick#3", 30);

Create creator = new Create("Creator", 15, example: cure1);

creator.SetTask(cure2);

creator.SetTask(cure3);

creator.SetCreatingChances(5, 1, 4);

model.SetNextElement(creator);

Process duty1 = new Process("DutyDoctorx#1", distribution: "exp", queueLimit: 0);

duty1.SetChooseCondition(SelectionCondition1);

Process duty2 = new Process("DutyDoctorx#2", distribution: "exp", queueLimit: 0);

duty2.SetChooseCondition(SelectionCondition1);

ProcessBlock dutyBase = new ProcessBlock("DutyBase", workers: new Process[] { duty1, duty2 });

dutyBase.SetQueuePickCondition(QueuePickConditional);

model.SetNextElement(dutyBase);

Counter firstCoridor = new Counter("Coridor#1", 2, 5, "unif");

firstCoridor.Delaying = true;

dutyBase.SetNext(firstCoridor);

model.AddElement(firstCoridor);

Process register = new Process("Register", 4.5, 3, "erl");

firstCoridor.SetNext(register);

model.AddElement(register);

Counter secondCoridor = new Counter("Coridor#2", 2, 5, "unif");

secondCoridor.Delaying = true;

register.SetNext(secondCoridor);

model.AddElement(secondCoridor);

Process lab1 = new Process("Laboratory#1", 4, 3, distribution: "erl", queueLimit: 0);

lab1.SetChooseCondition(SelectionCondition2);

Process lab2 = new Process("Laboratory#2", 4, 3, distribution: "erl", queueLimit: 0);

lab2.SetChooseCondition(SelectionCondition2);

ProcessBlock lab = new ProcessBlock("Laboratory", workers: new Process[] { lab1, lab2 });

secondCoridor.SetNext(lab);

lab.SetNext(dutyBase);

model.AddElement(lab);

Process convoy1 = new Process("Convoy#1", 3, 8, distribution: "unif", queueLimit: 0);

Process convoy2 = new Process("Convoy#2", 3, 8, distribution: "unif", queueLimit: 0);

Process convoy3 = new Process("Convoy#3", 3, 8, distribution: "unif", queueLimit: 0);

ProcessBlock convoy = new ProcessBlock("ConvoyBand", workers: new Process[] { convoy1, convoy2, convoy3 });

dutyBase.AddNext(convoy);

model.AddElement(convoy);

Counter final = new Counter("Exit");

lab.AddNext(final);

convoy.SetNext(final);

model.AddElement(final);

model.Simulate(1000);

Component SelectionCondition1(Task forSending, List<Component> components)

{

forSending.processingDelay = -1;

if (forSending.name.EndsWith('1'))

return components[1];

else

return components[0];

}

Component SelectionCondition2(Task forSending, List<Component> components)

{

if (forSending.name.EndsWith('3'))

return components[1];

else

{

forSending.name = "Sick#1";

forSending.processingDelay = 15;

return components[0];

}

}

int QueuePickConditional(List<Task> list)

{

for(int i = 0; i < list.Count; i++)

{

if (list[i].name.EndsWith('1'))

return i;

}

return 0;

}

}

}

}

По виконанню програми, складно, але можна побачити структуру системи:

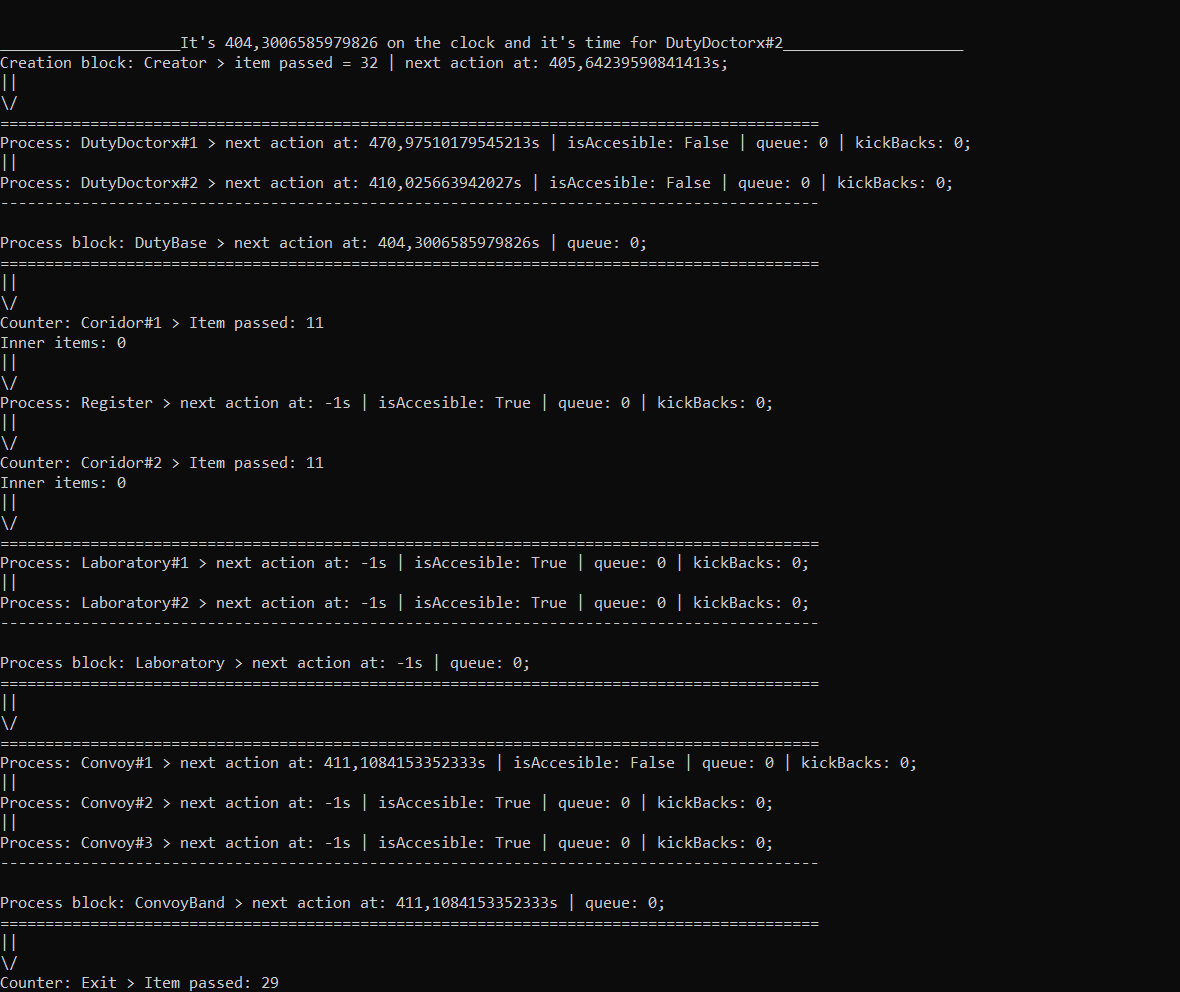


Рисунок 2.6 – Приклад роботи алгоритму (одна з ітерацій)

Щодо результатів одного із запуску програми:

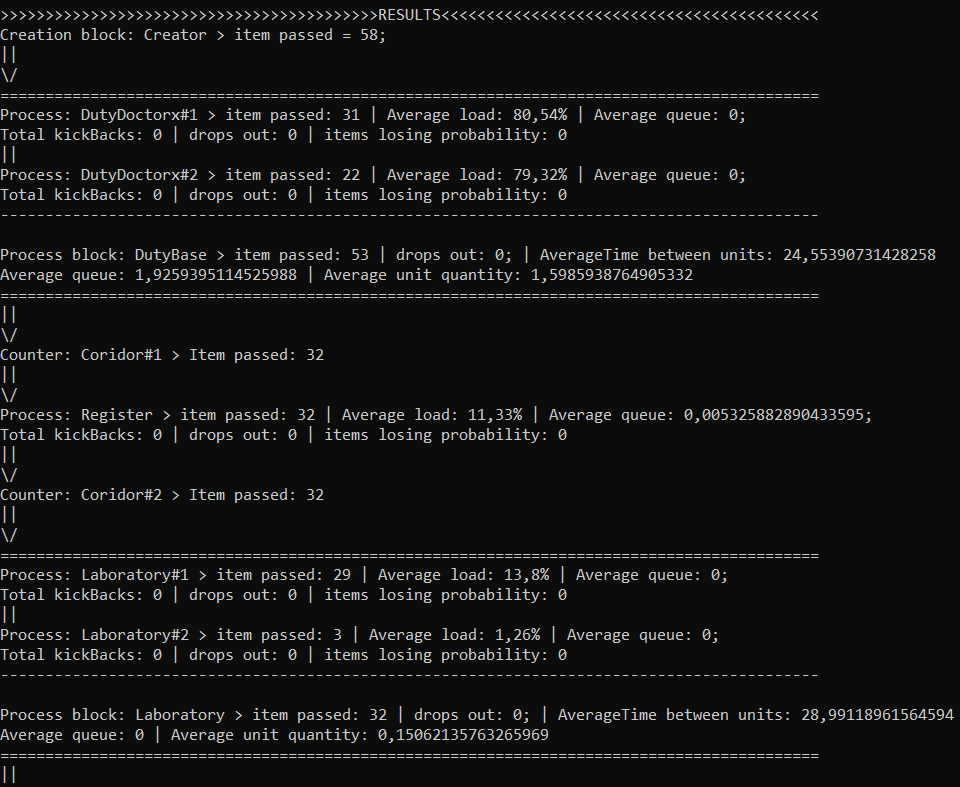


Рисунок 2.7 – Результати запуску (1 частина)

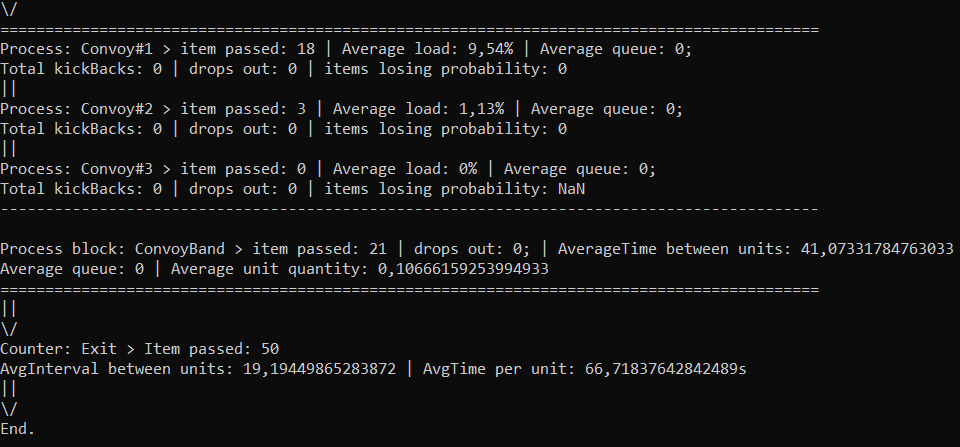


Рисунок 2.8 – Результати запуску (2 частина)

Тож, можна побачити, що:

1. Середній час хворого у лікарні: 66 хв
2. Середній інтервал надходження хворих до лабораторії: 28.9 хв

Висновок

В рамках даної лабораторної роботи згідно із завданням було реалізовано алгоритм побудови імітаційних моделей дискретно-подійних систем. Були ретельно досліджені можливі нюанси і складності виконання алгоритму із різними схемами систем. У зв’язку із цим, за допомогою об’єктно-орієнтованого підходу була створена система класів із надійними функціоналом та взаємодією.

Алгоритм був протестований на різних схемах систем, для перевірки виконання усіх зазначених у завданні вимог. Зокрема було перевірено коректність поведінки на схемі, вказаній у завданні, за використанням верифікації. У результаті чого, можна було спостерігати логічні зміни вихідних параметрів після підстановки різних вхідних значень системи.

У висновку, можу сказати, що створений алгоритм є доволі універсальним, за винятком декількох моментів. Його побудова потребувала не стільки часу, скільки архітектурних рішень із урахуванням явних та майбутніх потреб.