ФЕДЕРАЛЬНОЕ АГЕНСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ Императора Александра I»

Кафедра «Информационные и вычислительные системы»

Дисциплина «Моделирование систем»

**ОТЧЁТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4**

**Вариант №30**

|  |  |
| --- | --- |
| Выполнил студент  Факультет: АИТ  Группа: ИВБ-211 | Шефнер А. |
| Проверил:  Доцент кафедры  «Информационные и вычислительные  системы» | Гончаренко В. А. |

**Санкт-Петербург**

**2024**

**Цель работы:**

Целью лабораторного занятия является закрепление теоретических основ имитационного моделирования систем массового обслуживания и приобретение практических навыков разработки имитационных моделей типовых систем на языках высокого уровня.

**Исходные данные:**

H2 / M / 1

**Блок схема алгоритма:**



**Исходный текст программы:**

**Index.html**

<!DOCTYPE HTML>

<html>

<head>

<title>Моделирование систем</title>

<meta charset="UTF-8" />

<link rel="stylesheet" type="text/css" href="index.css">

</head>

<body>

<h1>Шефнер Альберт - лабораторная работа №4.</h1>

<h2>

Дисциплина "Моделирование систем"<br>

Кафедра "Информационные и вычислительные системы"

</h2>

<div>

<table>

<tr>

<td><p>Количество заявок</p></td>

<td><input id="jobInput" type="number" value="1000"/></td>

</tr>

<tr>

<td><p>Интенсивность нагрузки: параметр λ1</p></td>

<td><input id="loadFactorRate1Input" type="number" value="0.5"/></td>

</tr>

<tr>

<td><p>Интенсивность нагрузки: параметр λ2</p></td>

<td><input id="loadFactorRate2Input" type="number" value="0.5"/></td>

</tr>

<tr>

<td><p>Интенсивность нагрузки: параметр порога</p></td>

<td><input id="loadFactorThresholdInput" type="number" value="0.5"/></td>

</tr>

<tr>

<td><p>Интенсивность обработки</p></td>

<td><input id="processingFactorInput" type="number" value="0.5"/></td>

</tr>

<tr>

<td><p>Размер очереди</p></td>

<td><input id="maxQueueInput" type="number" value=""/></td>

</tr>

<tr>

<td><p>Предельное время моделирования</p></td>

<td><input id="maxTimeInput" type="number" value="1000"/></td>

</tr>

</table>

<div>

<button id="startButton" type="button"> Вычислить </button>

</div>

<div>

<h2>Результат моделирования</h2>

<p id="outputParagraph">Нажмите "Вычислить" для провередия расчётов</p>

</div>

</body>

<script src="index.js"> </script>

</html>

index.css

body, h1, h2, p {

margin: 0;

padding: 0;

}

body {

background-color: #f4f4f4;

font-family: Arial, sans-serif;

color: #333;

line-height: 1.6;

padding: 20px;

}

h1 {

text-align: center;

margin-bottom: 20px;

color: #2c3e50;

}

h2 {

text-align: center;

margin: 20px 0;

color: #2980b9;

}

table {

width: 100%;

max-width: 600px;

margin: 0 auto;

border-collapse: collapse;

background-color: #fff;

box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

}

td {

padding: 10px;

border: 1px solid #ddd;

}

input[type="number"] {

width: 100%;

padding: 8px;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;

}

button {

display: block;

width: 100%;

max-width: 200px;

margin: 20px auto;

padding: 10px;

background-color: #2980b9;

color: white;

border: none;

border-radius: 4px;

cursor: pointer;

font-size: 16px;

}

button:hover {

background-color: #3498db;

}

#outputParagraph {

text-align: center;

margin-top: 20px;

font-weight: bold;

}

**index.js**

// @ts-check

/\*

\* H2 / M / 1

\*/

/\*\*

\* @typedef Job

\* @prop arrivalTime {number}

\* @prob processingTime {number}

\* @prop waitTime {number}

\*/

/\*\*

\* @typedef QueueSample

\* @prop size {number}

\* @prop time {number}

\*/

/\*\*

\* @typedef ModellingParams

\* @prop maxQueue {number}

\* @prop maxTime {number}

\* @prop loadFactorRate1 {number}

\* @prop loadFactorRate2 {number}

\* @prop loadFactorThreshold {number}

\* @prop processingRate {number}

\* @prop jobCount {number}

\*/

/\*\*

\* @typedef ModellingStats

\* @prop averageStayingTime {number}

\* @prop rejectionFrequency {number}

\* @prop averageWaitTime {number}

\* @prop averageProcessingTime {number}

\* @prop variationCoefficient {number}

\*/

const jobInput = /\*\* @type {HTMLInputElement} \*/ (document.querySelector("#jobInput"));

const loadFactorRate1Input = /\*\* @type {HTMLInputElement} \*/

(document.querySelector("#loadFactorRate1Input"));

const loadFactorRate2Input = /\*\* @type {HTMLInputElement} \*/

(document.querySelector("#loadFactorRate2Input"));

const loadFactorThresholdInput = /\*\* @type {HTMLInputElement} \*/

(document.querySelector("#loadFactorThresholdInput"));

const processingFactorInput = /\*\* @type {HTMLInputElement} \*/

(document.querySelector("#processingFactorInput"));

const maxQueueInput = /\*\* @type {HTMLInputElement} \*/ (document.querySelector("#maxQueueInput"));

const maxTimeInput = /\*\* @type {HTMLInputElement} \*/ (document.querySelector("#maxTimeInput"));

const startButton = /\*\* @type {HTMLInputElement} \*/ (document.querySelector("#startButton"));

const outputParagraph = /\*\* @type {HTMLInputElement} \*/ (document.querySelector("#outputParagraph"))

/\*\*

\* @param rate {number}

\* @returns {number}

\*/

const exponentialRand = (rate) => (-1 / rate) \* Math.log(Math.random())

/\*\*

\* @param rate1 {number}

\* @param rate2 {number}

\* @param threshold {number}

\* @returns {number}

\*/

const hyperExponentialRand = (rate1, rate2, threshold) =>

exponentialRand(threshold > Math.random() ? rate1 : rate2)

/\*\*

\* @param array {number[]}

\* @returns {number}

\*/

const standardDeviation = (array) => {

const n = array.length

const mean = array.reduce((a, b) => a + b) / n

return Math.sqrt(array.map(x => Math.pow(x - mean, 2)).reduce((a, b) => a + b) / n)

}

/\*\*

\* @param params {ModellingParams}

\* @returns {ModellingStats}

\*/

const doModelling = (params) => {

let rejected = 0;

let processed = 0;

let modellingTime = 0;

let channelReleaseTime = 1 / 0;

let stayingTime = 0;

let queue = [];

let waitTimes = [];

let processingTimes = [];

do {

const arrivalTime = modellingTime + hyperExponentialRand(

params.loadFactorRate1,

params.loadFactorRate2,

params.loadFactorThreshold,

);

if (arrivalTime > channelReleaseTime) {

modellingTime = channelReleaseTime;

const currentProcessingTime = exponentialRand(params.processingRate);

processingTimes.push(currentProcessingTime);

channelReleaseTime = queue.length == 1

? channelReleaseTime = 1 / 0

: channelReleaseTime = modellingTime + currentProcessingTime;

console.assert(queue.length > 0);

stayingTime += (modellingTime - queue[0])

const waitTime = Math.max(0, modellingTime - queue[0] - currentProcessingTime);

waitTimes.push(waitTime);

queue.splice(0, 1);

processed++;

} else {

modellingTime = arrivalTime

if (queue.length == 0) {

const currentProcessingTime = exponentialRand(params.processingRate);

channelReleaseTime = modellingTime + currentProcessingTime;

}

if (queue.length >= params.maxQueue) {

rejected++;

} else {

queue.push(modellingTime);

}

}

} while (modellingTime < params.maxTime && processed + rejected < params.jobCount);

const rejectionFrequency = rejected / modellingTime;

const averageWaitTime = waitTimes.reduce((a, b) => a + b, 0) / waitTimes.length;

const averageProcessingTime = processingTimes.reduce((a, b) => a + b, 0) / processingTimes.length;

const averageStayingTime = isNaN(averageWaitTime) ? averageProcessingTime : stayingTime / processed;

const variationCoefficient = standardDeviation(processingTimes) / averageProcessingTime;

return {

averageStayingTime,

rejectionFrequency,

averageWaitTime,

averageProcessingTime,

variationCoefficient

}

}

const main = () => {

const loadFactorRate1 = parseFloat(loadFactorRate1Input.value);

const loadFactorRate2 = parseFloat(loadFactorRate2Input.value);

const loadFactorThreshold = parseFloat(loadFactorThresholdInput.value);

const processingRate = parseFloat(processingFactorInput.value);

const maxQueue = maxQueueInput.value ? parseFloat(maxQueueInput.value) : 1 / 0;

const maxTime = maxTimeInput.value ? parseFloat(maxTimeInput.value) : 1 / 0;

const jobCount = parseInt(jobInput.value);

/\*\* @type {ModellingParams} \*/

const params = {

maxQueue,

maxTime,

loadFactorRate1,

loadFactorRate2,

loadFactorThreshold,

processingRate,

jobCount,

};

const result = doModelling(params);

outputParagraph.innerText =

`Частота отказов: ${result.rejectionFrequency}\n` +

`Среднее время ожидания: ${result.averageWaitTime}\n` +

`Среднее время обслуживания: ${result.averageProcessingTime}\n` +

`Среднее время пребывания в системе: ${result.averageStayingTime}\n` +

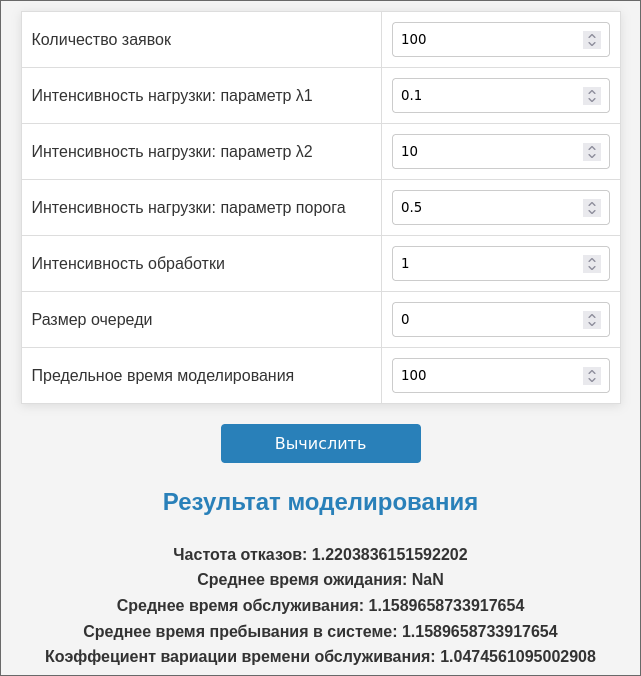
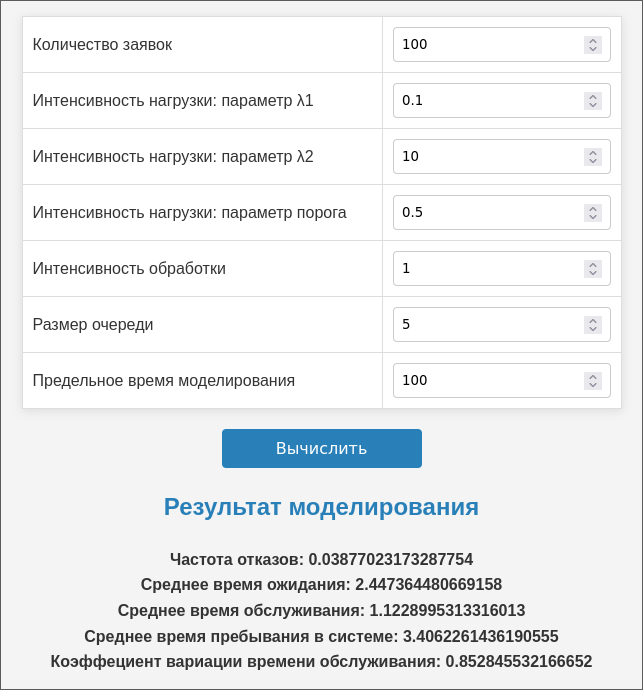
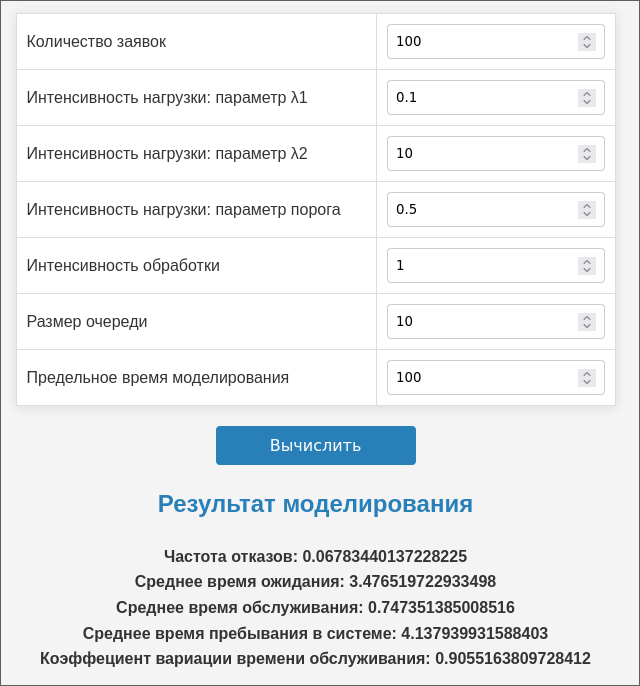
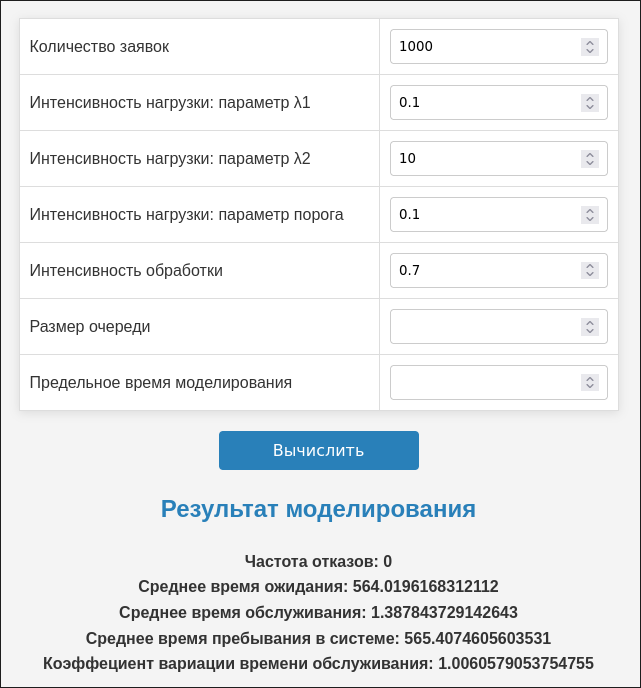
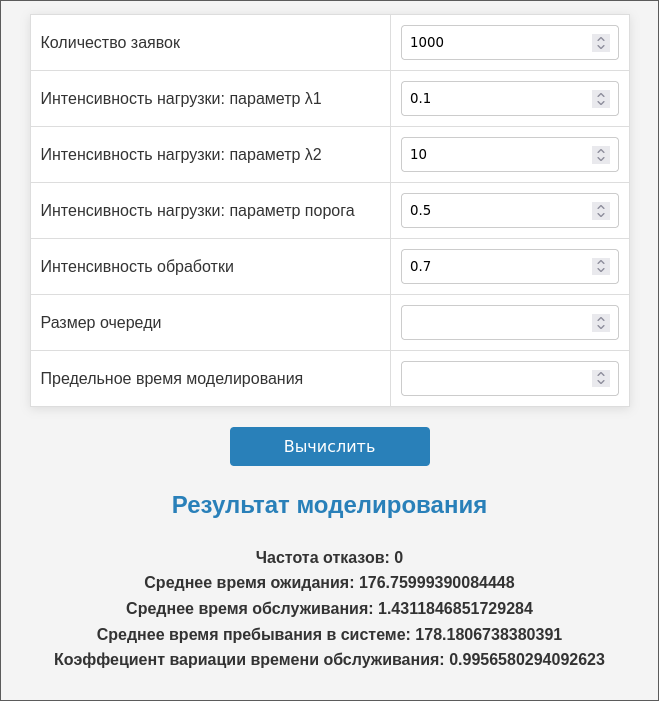
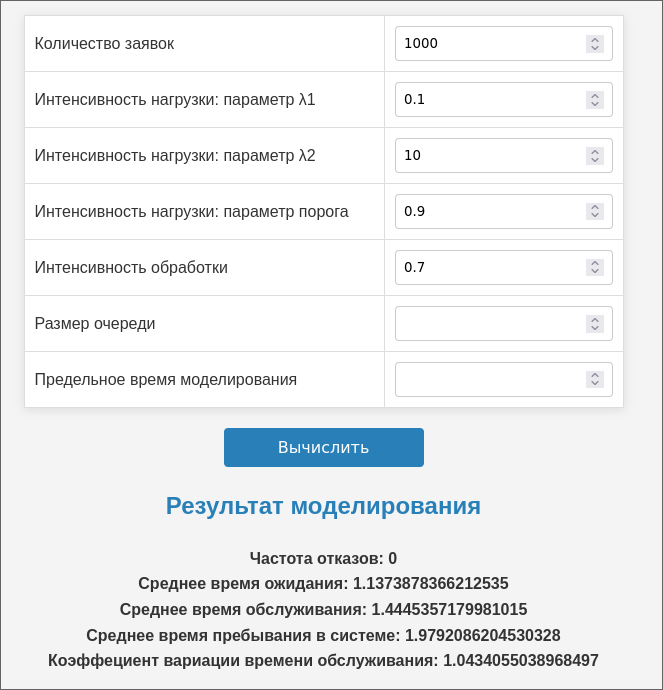
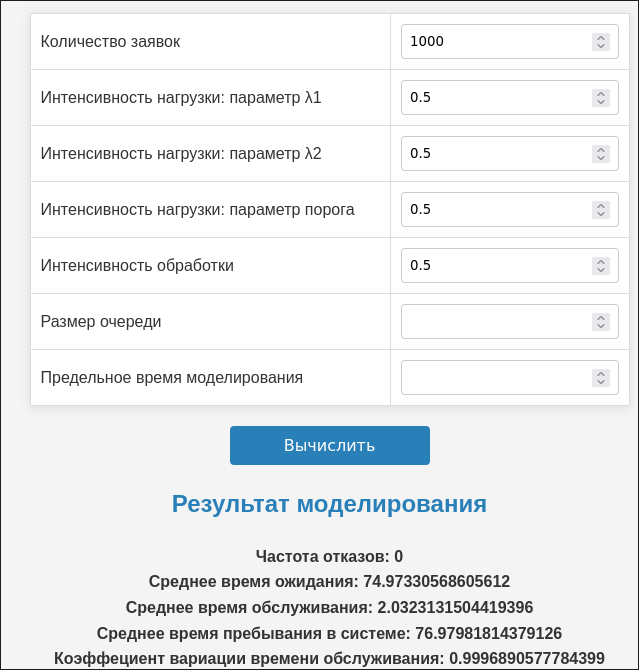
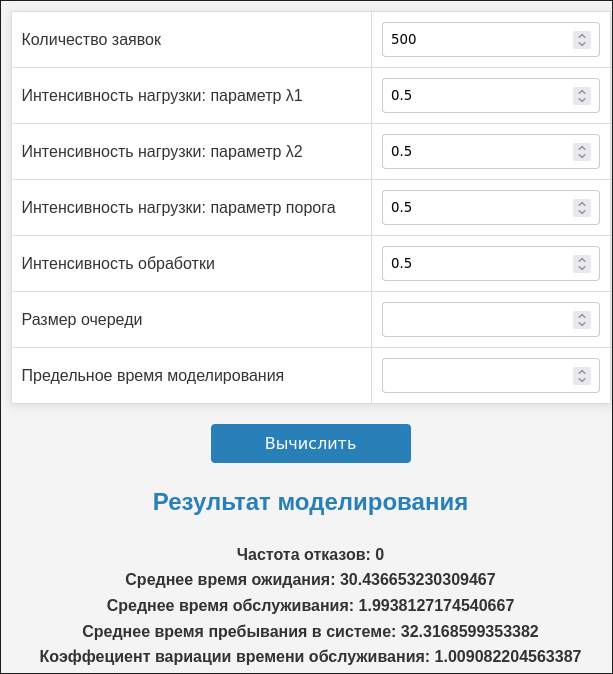
`Коэффециент вариации времени обслуживания: ${result.variationCoefficient}\n`

}

// @ts-ignore

startButton.onclick = main

**Результаты моделирования:**



**Вывод:**

Была разработана имитационная модель при помощи средств языка программирования JavaScript, языков разметки HTML и CSS. Проведены испытания модели и получены статистические характеристики модели. Имитационная модель позволяет наглядно увидеть как ведёт себя система, если вычислить её характеристики аналитически сложно или невозможно.