МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №6**

**Дисциплина: Основы компьютерного моделирования**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Курбатский В.А.

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Направленность (профиль): Компьютерные технологии и прикладная математика.

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кособуцкая Е. В.

Краснодар

2023

**1 Постановка задачи**

В вычислительном центре имеются две ЭВМ. Задания на обработку поступают каждые 2 минуты в пункт приема. Здесь в течении 12 минут они регистрируются и сортируются оператором, после первой обработки на ЭВМ обнаруживаются ошибки ввода, которые сразу же в течение 3 минут исправляются оператором. На время корректировки ввода задание не освобождает соответствующей ЭВМ, и после корректировки начинается его повторная обработка. Возможность ошибки при повторной обработке исключается, т. е. повторная обработка всегда является окончательной. Продолжительность работы ЭВМ при обработке задания в каждом случае составляет 10 минут. В центре имеется лишь одно рабочее место оператора.

Разработать программу, моделирующую процесс функционирования вычислительного центра при условии, что обработать необходимо 100 заданий.

Разработанная программа должна удовлетворять следующим требованиям:

* обеспечивать ввод исходных данных;
* поддерживать интерактивное редактирование;
* производить имитационное моделирование;
* представлять результаты моделирования в удобном виде;

**2 Описание разработанного алгоритма**

Алгоритм симуляции системы заключен в бесконечный цикл с постусловием, в начале каждой итерации алгоритм проверяет не равно ли время до поступления новой задачи в систему нулю, если да, задача то счетчик очереди и счетчик количества прибывших задач увеличивается на один, а также обновляется время до следующей задачи. Далее если в очереди есть задачи и оператор не работает, он начинает обрабатывать задачи, при этом время, присваивается время сортировки текущей задачи, количество задач в очереди уменьшается, а флаг сортировки задания переходит в рабочее положение. Далее проверяется закончил ли оператор первичную обработку, если да, обработанное задание перемещается в очередь машины с меньшей очередью, со значением 0, обозначающем, что задача обрабатывается на ЭВМ в первый раз. Далее проверяется, не закончил ли корректирование ошибок оператор, если да, то увеличивается счетчик времени, потраченного на обработку ошибок, и задание возвращается начало очереди соответствующей ЭВМ со значением 1 и удаляется из очереди корректировки ошибок. Далее проверяется, закончила ли первая машина вычисления, если да, счетчик времени вычислений первой машины увеличивается на величину выполнения одного задания и определяется ошибка, если ошибка возникла, машина останавливается, задание помещается в очередь на корректирование оператором со значением 1, если ошибка не возникла задание удаляется из очереди машины ,счетчик количества выполненных заданий и выполненных заданий на первой машине увеличиваются на 1, машина переходит в режим ожидания. Далее выполняется аналогичная проверка для второй машины. Потом проверяется, есть ли задания в очереди ошибок, если да, флаг сортировки оператором задач становится не активным, а флаг корректирования ошибок активным, счетчику времени корректирования текущего задания присваивается время корректирования задания. Далее проверяется, есть ли работает ли первая ЭВМ и является ли последние задание в очереди не разу не обработанным, если да, времени обработки текущего задания присваивается время вычисления задачи, а флаг активности ЭВМ переводится в активное состояние. Дальше следует аналогичная проверка для ЭВМ 2. После чего сверяется количество выполненных задач и задач которые необходимо выполнить, если они равны алгоритм завершает свою работу. Потом уменьшаются счетчики: времени сортировки, корректирования, вычисления при работе соответствующих флагов. Увеличивается общее время работы системы и время параллельной работы машин при активности соответствующих флагов.

После выхода из цикла вычисляются остальные параметры, необходимые для правильного восприятия работы системы.

**4 Тестовые примеры и результаты тестирования программы**

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – Результаты работы программы

**4 Текст разработанной программы**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace SimulationComputingCenter

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

groupBox2.Controls.Clear();

int timeToNextTask = Convert.ToInt32(textBox1.Text);

int timeSort = Convert.ToInt32(textBox2.Text);

int timeErrCorrection = Convert.ToInt32(textBox3.Text);

int ErrProbility = Convert.ToInt32(textBox4.Text);

int timeComputing = Convert.ToInt32(textBox5.Text);

int countTask = Convert.ToInt32(textBox6.Text);

Simulation model = new Simulation(countTask, timeToNextTask, timeSort, timeComputing, timeErrCorrection, ErrProbility);

Label result = new Label();

result.AutoSize = true;

result.Text = model.GetStringInfo();

result.Location = new Point(20, 20);

groupBox2.Controls.Add(result);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SimulationComputingCenter

{

public static class RandomExtension

{

public static bool NextBool(this Random rnd, int probility = 50)

{

bool[] bools = new bool[100];

for(int i = 0; i < 100; i++)

{

if(i <= probility)

{

bools[i] = true;

}

else

{

bools[i] = false;

}

}

return bools[rnd.Next(0, 100)];

}

}

}

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SimulationComputingCenter

{

public class Simulation

{

int countTask, timeToNextTask, timeSort, timeComputing, timeErrCorection;

int ErrProbility;

int totalTimeWorking = 0, operatorDownTime = 0, EMC1DownTime = 0, EMC2DownTime = 0, totalTimeComputingEMC1 = 0, totalTimeComputingEMC2 = 0;

int timeWorkingOperator = 0, totalSortTime = 0, totalTimeCorrectionErr = 0, countErr = 0, countTaskInQue = 0;

int timeCorrectionErrEMC1 = 0, timeCorrectionErrEMC2 = 0, countTaskComputingInEMC1 = 0, countTaskComputingInEMC2 = 0;

int parallelComputing = 0;

public Simulation(int countTask, int timeToNextTask, int timeSort, int timeComputing, int timeErrCorection, int ErrProbility)

{

this.countTask = countTask;

this.ErrProbility = ErrProbility;

this.timeSort = timeSort;

this.timeComputing = timeComputing;

this.timeErrCorection = timeErrCorection;

this.timeToNextTask = timeToNextTask;

Simulate();

}

void Simulate()

{

Random rnd = new Random();

int countExecuteTask = 0;

int timeBeforNextTask = 2;

int countTaskInQue = 0; //очередь на обработку оператором

int timeSorting = 0;

int timeCorrecting = 0;

bool operatorRepairErr = false; //оператор исправляет ошибки

bool operatorSorting = false; //оператор сортирует задачи

int timeComputingInECM1 = 0;

int timeComputingInECM2 = 0;

bool EMC1Working = false;

bool EMC2Working = false;

int countIncomingTasks = 0;

List<int> queErr = new List<int>(); // 1 - ЭВМ1, 2 - ЭВМ2

List<int> queEMC1 = new List<int>(); // 0 - не обработана, 1 - была обработана

List<int> queEMC2 = new List<int>(); // 0 - не обработана, 1 - была обработана

do

{

if (timeBeforNextTask == 0)

{

if (countIncomingTasks < this.countTask)

{

countTaskInQue++;

}

countIncomingTasks++;

timeBeforNextTask = this.timeToNextTask;

}

///

if (countTaskInQue > 0 && !operatorSorting && !operatorRepairErr)//начинает работать оператор

{

timeSorting = this.timeSort;

countTaskInQue--;

operatorSorting = true;

}

if (timeSorting == 0 && operatorSorting && !operatorRepairErr) //оператор закончил первичную обработку задачи

{

this.totalSortTime += this.timeSort;

operatorSorting = false;

if (queEMC1.Count <= queEMC2.Count)

{

queEMC1.Add(0);

}

else

{

queEMC2.Add(0);

}

}

if (timeCorrecting == 0 && operatorRepairErr)

{

this.totalTimeCorrectionErr += this.timeErrCorection;

if(queErr.First() == 1)

{

EMC1Working = true;

//Время коррекции ошибок первой машины

this.timeCorrectionErrEMC1 += this.timeErrCorection;

}

else

{

EMC2Working = true;

//Время коррекции ошибок второй машины

this.timeCorrectionErrEMC2 += this.timeErrCorection;

}

queErr.RemoveAt(0);

operatorRepairErr = false;

}

///

//if (countTaskInQue > 0 && !operatorSorting && !operatorRepairErr)//начинает работать оператор

//{

// timeSorting = this.timeSort;

// countTaskInQue--;

// operatorSorting = true;

//}

if (timeComputingInECM1 == 0 && EMC1Working) //вычислила задачу первая машина

{

if (queEMC1.First() == 0)

{

if (rnd.NextBool(this.ErrProbility))

{

timeComputingInECM1 += this.timeComputing;

EMC1Working = false;

queErr.Add(1);

//operatorRepairErr = true;

queEMC1[0] = 1;

}

else

{

queEMC1.RemoveAt(0);

countExecuteTask++;

EMC1Working = false;

//Увеличиваем количество задач выполненных на ЭВМ1

this.countTaskComputingInEMC1++;

}

}

else

{

if (queEMC1.First() == 1)

{

queEMC1.RemoveAt(0);

countExecuteTask++;

EMC1Working = false;

//Увеличиваем количество задач выполненных на ЭВМ1

this.countTaskComputingInEMC1++;

}

}

this.totalTimeComputingEMC1 += this.timeComputing;

}

if (timeComputingInECM2 == 0 && EMC2Working) //вычислила задачу первая машина

{

if (queEMC2.First() == 0)

{

if (rnd.NextBool(this.ErrProbility))

{

timeComputingInECM2 += this.timeComputing;

EMC2Working = false;

queErr.Add(2);

//operatorRepairErr = true;

queEMC2[0] = 1;

}

else

{

queEMC2.RemoveAt(0);

countExecuteTask++;

EMC2Working = false;

//Увеличиваем количество задач выполненных на ЭВМ2

this.countTaskComputingInEMC2++;

}

}

else

{

if (queEMC2.First() == 1)

{

queEMC2.RemoveAt(0);

countExecuteTask++;

EMC2Working = false;

//Увеличиваем количество задач выполненных на ЭВМ2

this.countTaskComputingInEMC2++;

}

}

this.totalTimeComputingEMC2 += this.timeComputing;

}

if(queErr.Count > 0 && !operatorRepairErr)

{

timeCorrecting += this.timeErrCorection;

operatorRepairErr = true;

}

if (queEMC1.Count > 0 && !EMC1Working && queEMC1.First() == 0) // машина не работает - пусть работает

{

timeComputingInECM1 = this.timeComputing;

EMC1Working = true;

}

if (queEMC2.Count > 0 && !EMC2Working && queEMC2.First() == 0)// ты тоже работай

{

timeComputingInECM2 = this.timeComputing;

EMC2Working = true;

}

if (countExecuteTask == countTask)

{

break;

}

this.totalTimeWorking++;

timeBeforNextTask--;

if (operatorSorting && !operatorRepairErr)

{

timeSorting--;

}

if (operatorRepairErr)

{

timeCorrecting--;

}

if (EMC1Working)

{

timeComputingInECM1--;

}

if (EMC2Working)

{

timeComputingInECM2--;

}

if (EMC1Working && EMC2Working)

{

parallelComputing++;

}

} while (true);

this.EMC1DownTime = this.totalTimeWorking - this.totalTimeComputingEMC1;

this.EMC2DownTime = this.totalTimeWorking - this.totalTimeComputingEMC2;

this.timeWorkingOperator = this.totalTimeCorrectionErr + this.totalSortTime;

this.operatorDownTime = this.totalTimeWorking - this.timeWorkingOperator;

this.countErr = this.totalTimeCorrectionErr / this.timeErrCorection;

this.countTaskInQue = countIncomingTasks - this.countTask;

}

public string GetStringInfo()

{

string info = $"Общее время работы: {this.totalTimeWorking} минут\n"

+ $"Время работы ЭВМ1: {this.totalTimeComputingEMC1} минут\n"

+ $"Время простоя ЭВМ1: {Math.Round(this.EMC1DownTime \* 100.0 / this.totalTimeWorking, 2)} %\n"

+ $"Время работы ЭВМ2: {this.totalTimeComputingEMC2} минут\n"

+ $"Время простоя ЭВМ2: {Math.Round(this.EMC2DownTime \* 100.0 / this.totalTimeWorking, 2)} %\n"

+ $"Интенсивность нагрузки: {Math.Round(this.countTaskInQue \* 1.0 / this.countTask, 2)}\n"

+ $"Время работы оператора: {this.timeWorkingOperator} минут\n"

+ $"Время коррекции ошибок оператором: {this.totalTimeCorrectionErr} минут\n"

+ $"Время регистрации и сортировки задач оператором: {this.totalSortTime} минут\n"

+ $"Количество заданий в очереди: {this.countTaskInQue} штук\n"

+ $"Пропускная способность: {Math.Round(this.countTask \* 1.0 / this.totalTimeWorking, 2)} задач/минута\n"

+ $"Средние время в очереди: {this.timeSort + Math.Round(this.totalTimeCorrectionErr \* 1.0 / this.countTask, 2)} минут\n"

+ $"Количество ошибок обнаруженных ЭВМ1: {this.timeCorrectionErrEMC1 / this.timeErrCorection} штук\n"

+ $"Количество ошибок обнаруженных ЭВМ2: {this.timeCorrectionErrEMC2 / this.timeErrCorection} штук\n"

+ $"Количество задач обработаных ЭВМ1: {this.countTaskComputingInEMC1} штук\n"

+ $"Количество задач обработаных ЭВМ2: {this.countTaskComputingInEMC2} штук\n"

+ $"Среднее время выполнения задания: {Math.Round((this.totalTimeComputingEMC1 + this.totalTimeComputingEMC2 + this.totalTimeCorrectionErr) \* 1.0 / this.countTask, 2)} минут\n"

+ $"Среднее количество задействованых каналов: {Math.Round(this.parallelComputing \* 1.0 / ((this.countTask \* this.timeComputing + this.countErr \* this.timeComputing) / 2.0) \* 2.0, 2)} штук\n"

+ $"Среднее число заявок в очереди: {Math.Round(1.0 \* (this.countTask + this.countTaskInQue) / this.totalTimeWorking, 2)} штук/минута\n";

return info;

}

}

}