МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

Направление подготовки: «Программная инженерия»

**ОТЧЕТ**

по лабораторной работе

**Имитационное моделирование системы обслуживания потока заданий на ЭВМ (очереди)**

Выполнил:

студент группы 3822Б1ПР2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ермолаев В.А.

Подпись

Нижний Новгород

2023 г.

Содержание

[1. Введение 3](#_Toc153312379)

[2. Постановка задачи 4](#_Toc153312380)

[3. Руководство пользователя 5](#_Toc153312381)

[4. Руководство программиста 7](#_Toc153312382)

[4.1. Описание структуры программы 7](#_Toc153312383)

[4.2. Описание структур данных 7](#_Toc153312384)

[4.2.1. Queue (шаблонный класс) 7](#_Toc153312385)

[4.2.2. ServiceTypes.cpp 7](#_Toc153312386)

[5. Эксперименты 9](#_Toc153312387)

[6. Заключение 10](#_Toc153312388)

[7. Литература 11](#_Toc153312389)

[8. Приложение 1 12](#_Toc153312390)

[9. Приложение 2 13](#_Toc153312391)

[10. Приложение 3 15](#_Toc153312392)

[11. Приложение 4 16](#_Toc153312393)

[12. Приложение 5 17](#_Toc153312394)

[13. Приложение 6 19](#_Toc153312395)

[14. Приложение 7 20](#_Toc153312396)

[15. Приложение 8 32](#_Toc153312397)

1. Введение

Лабораторная работа посвящена изучению динамической структуры данных очередь. В ходе работы рассматриваются различные варианты хранения очереди, а также разрабатываются методы и программы решения задач с использованием очередей.

Очередь - это структура данных, в которой элементы хранятся в порядке их поступления. Вставка новых элементов производится в конец очереди, а извлечение - из начала. Подобная организация данных широко встречается в различных приложениях, например, в системах массового обслуживания, в операционной системе и т.д.

В качестве примера использования очереди в лабораторной работе предлагается задача разработки системы имитации однопроцессорной ЭВМ. В рамках лабораторной работы рассматривается одна из наиболее простых моделей обслуживания заданий в вычислительной системе. Эта модель обеспечивает лишь начальное ознакомление с проблемами моделирования и анализа эффективности функционирования реальных вычислительных систем.

1. Постановка задачи

Для вычислительной системы (ВС) требуется разработать программную систему для имитации процесса обслуживания заданий. При построении модели функционирования ВС должны учитываться следующие основные моменты обслуживания заданий:

* генерация нового задания;
* постановка задания в очередь для ожидания момента освобождения процессора;
* выборка задания из очереди при освобождении процессора после обслуживания очередного задания.

По результатам проводимых вычислительных экспериментов система имитации должна выводить информацию об условиях проведения эксперимента (интенсивность потока заданий, размер очереди заданий, производительность процессора, число тактов имитации) и полученные в результате имитации показатели функционирования вычислительной системы, в т.ч.

* количество поступивших в ВС заданий;
* количество тактов бездействия;
* среднее количество занятых процессоров;
* количество выполняемых задач;
* количество задач в очереди;
* количество выполненных задач.

Показатели функционирования вычислительной системы, получаемые при помощи систем имитации, могут использоваться для оценки эффективности применения ВС; по результатам анализа показателей могут быть приняты рекомендации о целесообразной модернизации характеристик ВС (например, при длительных простоях процессора и при отсутствии отказов от обслуживания заданий желательно повышение интенсивности потока обслуживаемых заданий и т.д.).

1. Руководство пользователя

При запуске программы необходимо параметры эмуляции (также присутствуют настройки по умолчанию). После нажать на кнопку «Старт».

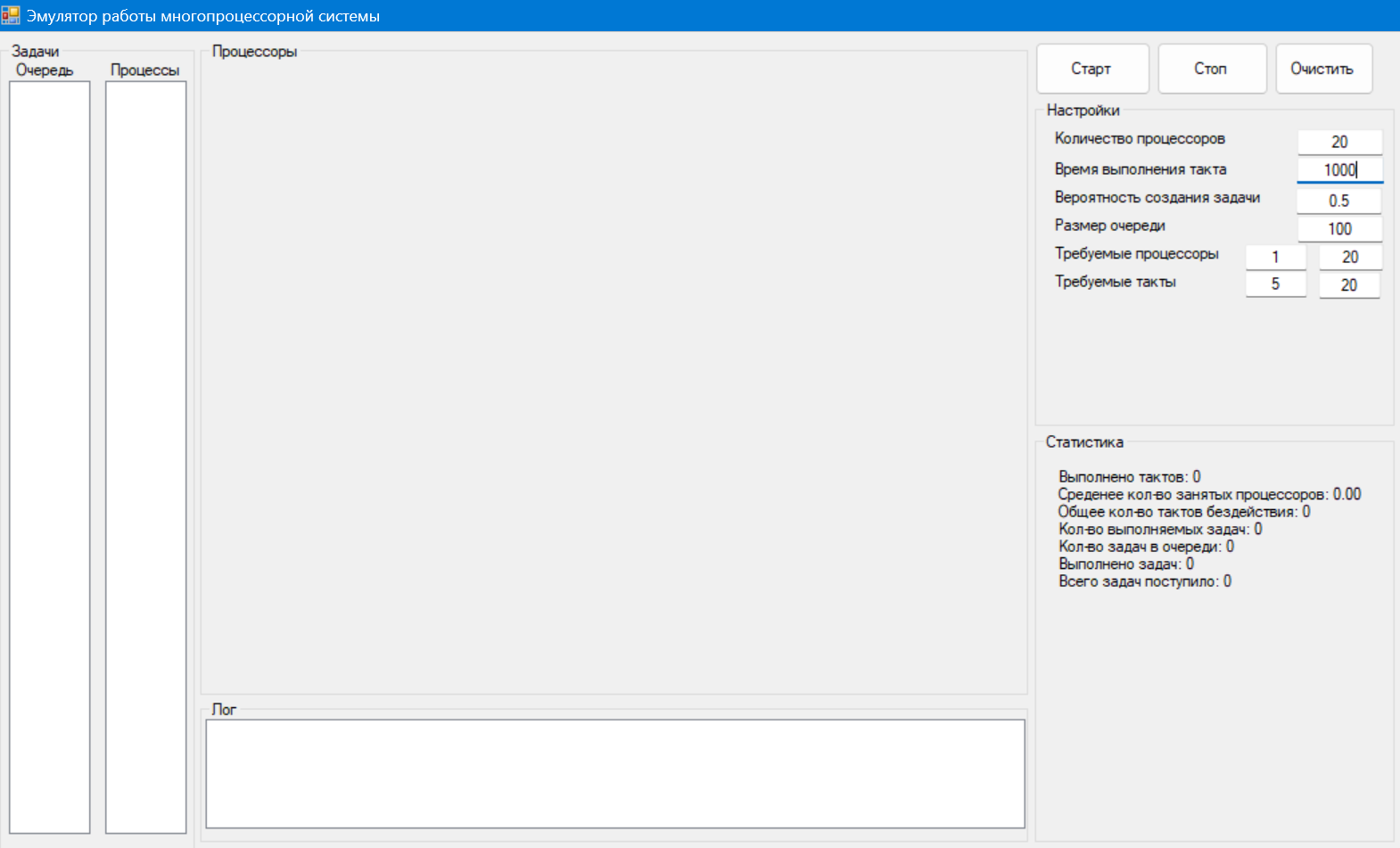


Рис. 1 – Исходное состояние

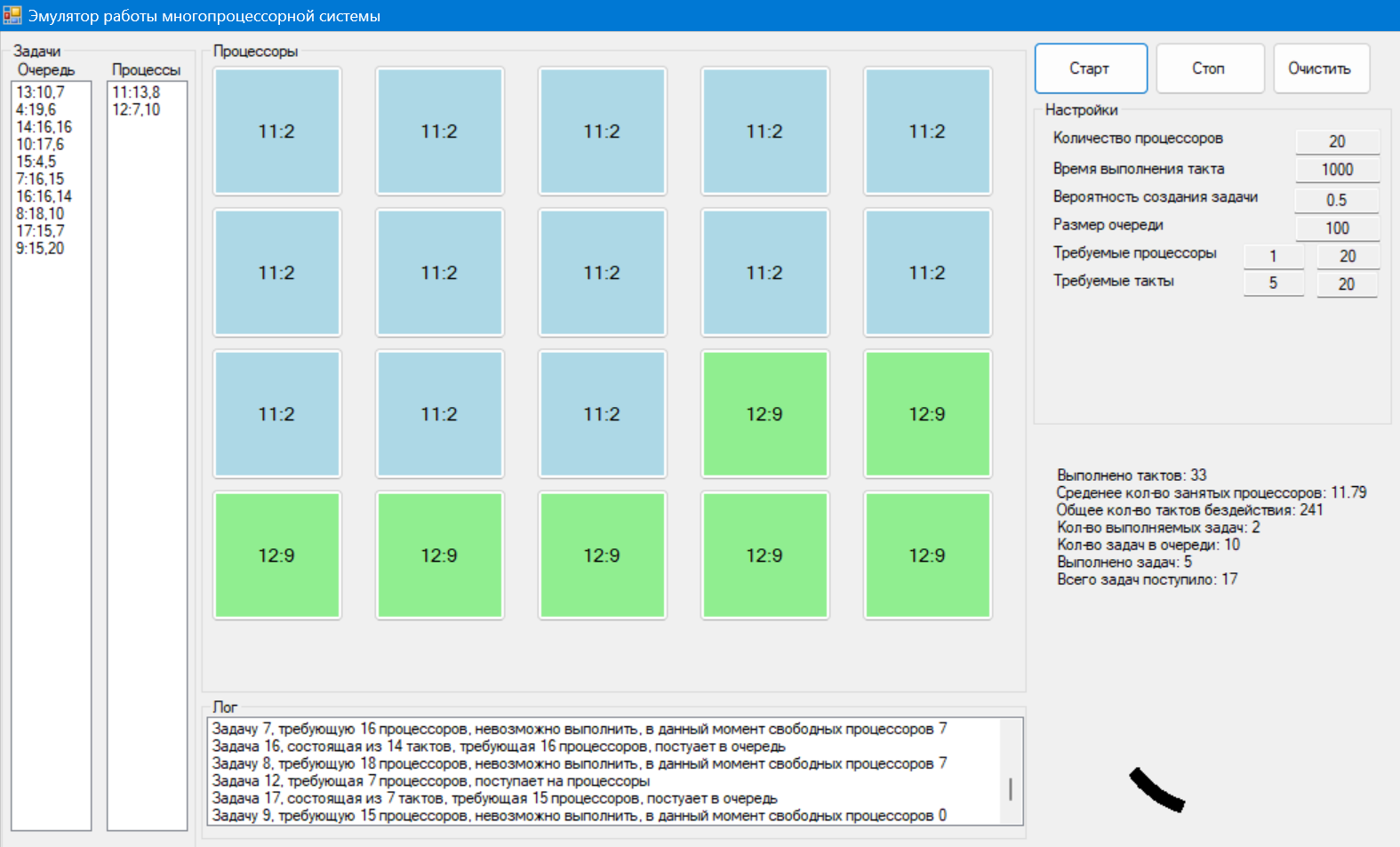


Рис. 2 –Рабочее состояние

Кнопка «Стоп» останавливает выполение программы до тех пор, пока не будет нажата кнопка «Старт».

Кнопка «Очистить» возвращает программу к исходному состоянию.

Круговая диаграмма в правом нижнем углу отражает заполненность кольцевой очерди.

1. Руководство программиста

## Описание структуры программы

Программа состоит из следующих модулей:

* Модуль Queue, содержащий реализацию класса Queue (файлы в директориях ./include/ и ./src/).
* Модуль Queue\_test. Набор тестов для класса Queue. Включает в себя файл ./test/test\_queue.cpp. Реализованы они с помощью использования фреймворка Google Test.
* Примеры использования класса Queue (файлы в директории ./samples/).
* Графическое приложение Queue\_gui (файлы в директории ./gui/)

## Описание структур данных

### Queue (шаблонный класс)

protected:

size\_t size – размер очереди

size\_t ind – индекс первого элемента

size\_t end – индекс конца очереди

size\_t count – количество элементов

T\* mas – массив размера size

public:

TQueue(size\_t n = 1) - конструктор

TQueue(TQueue <T>& queue) noexcept – конструтор копирования

~TQueue() noexcept – деструктор

void Push(T element) – добавить элемент в очередь

T Get() – изъять элемент из очереди

size\_t GetCount() noexcept – получить количество элементов

size\_t GetSize() noexcept – получить размер очереди

size\_t GetInd() noexcept – получить индекс первого элемнта

size\_t GetEnd() noexcept – получить индекс конца очереди

bool IsFull() noexcept – проверка на заполненность

bool IsEmpty() noexcept – проверка на пустоту

T& operator[](size\_t ind) noexcept – оператор индексации

const T& operator[](size\_t ind) const noexcept – константный оператор индексации

T& at(size\_t ind) – индексация с проверкой

const T& at(size\_t ind) const – константная индексация с проверкой

TQueue<T>& operator=(const TQueue<T>& stack) noexcept – оператор присваивания

friend std::ostream& operator<<(std::ostream& ostr, const TQueue<T>& queue) noexcept – оператор вывода

friend std::istream& operator>>(std::istream& istr, const TQueue<T>& queue) noexcept – оператор ввода

### ServiceTypes.cpp

struct Statistic {

uint64\_t tactCount = 0; - число совершенных тактов

double middleBusyProcessor = 0; - среднее число загруженных процессоров

uint64\_t tactInactionCount = 0; - число тактов бездействия

uint64\_t taskComplitedCount = 0; - количество завершонных задач

uint64\_t taskInWorkCount = 0; - количество задач в работе

uint64\_t taskSkipedCount = 0; - количество пропущенных задач

uint64\_t taskCount = 0; - количество поступивших задач

};

struct Task {

int id; - индивидульный номер задачи

size\_t tactCount; - требуемое число тактов для завершения задачи

size\_t procCount; - требуемое число процессоров

size\_t color; - цвет задачи

};

struct Processor {

Task task; - задача

size\_t stepCount = 0; -

size\_t taskTactCount = 0; -

size\_t emptyTactCount = 0; - количество тактов бездействия

bool isEmpty = true; - флаг занятости процессора

};

1. Эксперименты

Эксперименты проводились на ПК с следующими параметрами:

1. Операционная система: Windows 11
2. Процессор: Intel(R) Core™ i3-1115G4- CPU @ 3.00 GHz
3. Версия Visual Studio: 2022

Эксперименты показали, что добавление и изъятие из очереди происходят за константое время, т.е. сложность алгоримов – .

1. Заключение

Был реализован класс TQueue – кольцевая очередь с эффективными алгоритмами измения.

Было реализовано приложение, с использованием WinForms C++ - эмуляторсистемы обслуживания потока заданий на ЭВМ.

1. Литература
   * + 1. Лабораторный практикум. Составители: Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 105с.
2. Приложение 1

Queue.h

#ifndef \_MY\_QUEUE\_

#define \_MY\_QUEUE\_

#include <iostream>

template <class T>

class TQueue {

protected:

size\_t size;

size\_t ind;

size\_t end;

size\_t count;

T\* mas;

public:

TQueue(size\_t n = 1);

TQueue(TQueue <T>& queue) noexcept;

~TQueue() noexcept;

void Push(T element);

T Get();

size\_t GetCount() noexcept { return this->count; }

size\_t GetSize() noexcept { return this->size; }

size\_t GetInd() noexcept { return this->ind; }

size\_t GetEnd() noexcept { return this->end; }

bool IsFull() noexcept;

bool IsEmpty() noexcept;

T& operator[](size\_t ind) noexcept;

const T& operator[](size\_t ind) const noexcept;

T& at(size\_t ind);

const T& at(size\_t ind) const;

TQueue<T>& operator=(const TQueue<T>& stack) noexcept;

//îïåðàòîðû ââîäû è âûâîäû

friend std::ostream& operator<<(std::ostream& ostr, const TQueue<T>& queue) noexcept {

for (size\_t i = 0; i < queue.size; i++) {

ostr << i + 1 << ". " << queue.mas[i] << std::endl;

}

return ostr;

}

friend std::istream& operator>>(std::istream& istr, const TQueue<T>& queue) noexcept {

for (size\_t i = 0; i < queue.size; i++) {

istr >> queue.mas[i];

}

return istr;

}

};

#include "../src/MyQueue.hpp"

#endif

1. Приложение 2

MyQueue.hpp

#include "MyQueue.h"

template<class T>

TQueue<T>::TQueue(size\_t n): size(n)

{

if (size == 0)

throw std::domain\_error("Queue size = 0 throws Heap Curruption Error");

this->ind = 0;

this->end = this->size;

this->count = 0;

this->mas = new T[this->size]();

}

template<class T>

TQueue<T>::TQueue(TQueue<T>& q) noexcept

{

this->ind = q.ind;

this->end = q.end;

this->count = q.count;

this->size = q.size;

this->mas = new T[this->size];

std::copy(q.mas, q.mas + q.size, this->mas);

}

template<class T>

TQueue<T>::~TQueue() noexcept

{

if (this->mas == nullptr)

delete[] this->mas;

this->mas = nullptr;

this->count = 0;

this->end = 0;

this->ind = 0;

}

template<class T>

void TQueue<T>::Push(T element)

{

if (this->IsFull())

throw std::logic\_error("Queue is full");

size\_t index = (this->ind + this->count++) % this->size;

this->mas[index] = element;

this->end = (this->end + 1) % this->size;

}

template<class T>

T TQueue<T>::Get()

{

if (this->IsEmpty())

throw std::logic\_error("Queue is empty");

T tmp = this->mas[this->ind];

this->mas[this->ind] = T();

this->ind = (this->ind + 1) % this->size;

this->count--;

this->end = (this->end + 1) % this->size;

return tmp;

}

template<class T>

bool TQueue<T>::IsFull() noexcept

{

return this->count == this->size;

}

template<class T>

bool TQueue<T>::IsEmpty() noexcept

{

return this->count == 0;

}

// èíäåêñàöèÿ

template<class T>

T& TQueue<T>::operator[](size\_t ind) noexcept

{

return mas[(this->ind + ind) % this->size];

}

template<class T>

const T& TQueue<T>::operator[](size\_t ind) const noexcept

{

return mas[(this->ind + ind) % this->size];

}

// èíäåêñàöèÿ ñ êîíòðîëåì

template<class T>

T& TQueue<T>::at(size\_t ind)

{

if (ind >= this->size)

throw std::out\_of\_range("Index shoud be less then size");

return mas[(this->ind + ind) % this->size];

}

template<class T>

const T& TQueue<T>::at(size\_t ind) const

{

if (ind >= this->size)

throw std::out\_of\_range("Index shoud be less then size");

return mas[(this->ind + ind) % this->size];

}

template<class T>

TQueue<T>& TQueue<T>::operator=(const TQueue<T>& queue) noexcept

{

if (this == &queue)

return \*this;

if (this->size != queue.size) {

if (this->mas != nullptr)

delete[] this->mas;

this->size = queue.size;

this->mas = new T[this->size]();

}

std::copy(queue.mas, queue.mas + queue.size, this->mas);

this->count = queue.count;

this->ind = queue.ind;

this->end = queue.end;

return \*this;

}

1. Приложение 3

Queue\_sample.cpp

﻿#include <iostream>

#include "MyQueue.h"

using namespace std;

int main(int argc, char\*\* argv) {

setlocale(LC\_ALL, "Russian");

size\_t size, op;

int el;

try {

cout << "Введите размер очереди: ";

cin >> size;

TQueue<int> q(size);

while (true) {

cout << "Меню операций:\n"

<< "1. Добавить в очередь\n"

<< "2. Удалить элемент\n"

<< "3. Количество элементов\n"

<< "4. Размер\n"

<< "5. Начало\n"

<< "6. Конец\n"

<< "Ввести операцию: ";

cin >> op;

cout << endl;

switch (op) {

case 1:

cout << "Введите элемент (int): ";

cin >> el;

q.Push(el);

break;

case 2:

cout << q.Get() << endl;

break;

case 3:

cout << q.GetCount() << endl;

break;

case 4:

cout << q.GetSize() << endl;

break;

case 5:

cout << q.GetInd() << endl;

break;

case 6:

cout << q.GetEnd() << endl;

break;

default:

break;

}

cout << q << endl;

}

}

catch (exception e) {

cerr << e.what() << endl;

}

}

1. Приложение 4

test\_main.cpp

#include <gtest.h>

int main(int argc, char \*\*argv) {

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

1. Приложение 5

test\_polinom.cpp

#include "MyQueue.h"

#include <gtest.h>

TEST(TQueue, cant\_create\_with\_zero\_size)

{

ASSERT\_ANY\_THROW(TQueue<int> s1(0));

}

TEST(TQueue, can\_create)

{

ASSERT\_NO\_THROW(TQueue<int> s1(10));

}

TEST(TQueue, can\_get\_size)

{

TQueue<int> s1(2);

ASSERT\_EQ(2, s1.GetSize());

}

TEST(TQueue, can\_get\_ind)

{

TQueue<int> s1(2);

ASSERT\_EQ(0, s1.GetInd());

}

TEST(TQueue, can\_get\_end)

{

TQueue<int> s1(2);

ASSERT\_EQ(2, s1.GetEnd());

}

TEST(TQueue, can\_get\_count)

{

TQueue<int> s1(2);

ASSERT\_EQ(0, s1.GetCount());

}

TEST(TQueue, can\_push\_element)

{

TQueue<int> s1(1);

ASSERT\_NO\_THROW(s1.Push(10));

}

TEST(TQueue, cant\_push\_element\_when\_queue\_is\_full)

{

TQueue<int> s1(1);

s1.Push(10);

ASSERT\_ANY\_THROW(s1.Push(10));

}

TEST(TQueue, can\_pop\_element)

{

TQueue<int> s1(1);

int d;

s1.Push(10);

ASSERT\_NO\_THROW(d = s1.Get());

EXPECT\_EQ(10, d);

}

TEST(TQueue, cant\_pop\_element\_when\_queue\_is\_empty)

{

TQueue<int> s1(1);

ASSERT\_ANY\_THROW(s1.Get());

}

TEST(TQueue, can\_check\_is\_empty)

{

TQueue<int> s1(2);

ASSERT\_EQ(true, s1.IsEmpty());

s1.Push(1);

ASSERT\_EQ(false, s1.IsEmpty());

}

TEST(TQueue, can\_check\_is\_full)

{

TQueue<int> s1(1);

ASSERT\_EQ(false, s1.IsFull());

s1.Push(1);

ASSERT\_EQ(true, s1.IsFull());

}

TEST(TQueue, can\_assign\_queue\_to\_itself)

{

TQueue<int> s1(2);

ASSERT\_NO\_THROW(s1 = s1);

}

TEST(TQueue, can\_assign\_queues\_with\_different\_sizes)

{

TQueue<int> s1(2), s2(3);

ASSERT\_NO\_THROW(s1 = s2);

}

TEST(TQueue, can\_indexing)

{

TQueue<int> s1(2);

s1.Push(3);

s1.Push(4);

EXPECT\_EQ(3, s1[0]);

s1.Get();

EXPECT\_EQ(4, s1[0]);

s1.Push(5);

EXPECT\_EQ(5, s1[1]);

}

TEST(TQueue, can\_controlled\_indexing)

{

TQueue<int> s1(2);

ASSERT\_NO\_THROW(s1.at(0));

ASSERT\_ANY\_THROW(s1.at(3));

s1.Push(3);

s1.Push(4);

EXPECT\_EQ(3, s1.at(0));

s1.Get();

EXPECT\_EQ(4, s1.at(0));

s1.Push(5);

EXPECT\_EQ(5, s1.at(1));

}

1. Приложение 6

MyForm.cpp

#include "MyForm.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThread]

void Main(array<String^>^ args)

{

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

CppWinForm1::MyForm form;

Application::Run(% form);

}

1. Приложение 7

MyForm.h

﻿#pragma once

#include <vector>

#include <string>

#include <random>

#include <ctime>

#include <format>

#include <cmath>

#include "MyQueue.h"

#include "ServiceTypes.h"

#include <set>

namespace CppWinForm1 {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

public ref class MyForm : public System::Windows::Forms::Form

{

private: const size\_t TABLE\_COLUMS = 5;

public:

MyForm(void)

{

InitializeComponent();

this->statistic = new Statistic();

this->processors = new std::vector<Processor>;

this->taskQueue = new TQueue<Task>(queueSize);

this->colors = gcnew array<Color>(10) { Color::LightPink, Color::LightBlue, Color::LightGreen, Color::LightCoral,

Color::LightCyan, Color::LightGoldenrodYellow, Color::LightSalmon, Color::LightSeaGreen, Color::LightSkyBlue, Color::LightYellow };

srand(time(0));

updateStatistic();

}

protected:

~MyForm()

{

if (components)

{

delete components;

}

}

#pragma region Program Variable

private: std::vector<Processor>\* processors;

private: TQueue<Task>\* taskQueue;

private: array<Color>^ colors;

private: size\_t processorCount;

private: size\_t tactTime;

private: double taskProbability;

private: size\_t minRequiredProcessor;

private: size\_t maxRequiredProcessor;

private: size\_t minRequiredTact;

private: size\_t maxRequiredTact;

private: size\_t queueSize = 1;

private: Statistic\* statistic;

#pragma endregion

#pragma region WinForm components

private: System::Windows::Forms::GroupBox^ groupBox1;

private: System::Windows::Forms::GroupBox^ groupBox2;

private: System::Windows::Forms::GroupBox^ groupBox3;

private: System::Windows::Forms::GroupBox^ groupBox4;

private: System::Windows::Forms::GroupBox^ groupBox5;

private: System::Windows::Forms::Button^ button1;

private: System::Windows::Forms::Button^ button2;

private: System::Windows::Forms::Button^ button3;

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::Label^ label3;

private: System::Windows::Forms::Label^ label2;

private: System::Windows::Forms::Label^ label4;

private: System::Windows::Forms::Label^ label5;

private: System::Windows::Forms::Label^ label6;

private: System::Windows::Forms::Label^ label7;

private: System::Windows::Forms::Label^ label8;

private: System::Windows::Forms::Label^ label9;

private: System::Windows::Forms::Timer^ timer1;

private: System::Windows::Forms::Button^ button4;

private: System::Windows::Forms::TextBox^ textBox1;

private: System::Windows::Forms::TextBox^ textBox2;

private: System::Windows::Forms::TextBox^ textBox3;

private: System::Windows::Forms::TextBox^ textBox4;

private: System::Windows::Forms::TextBox^ textBox5;

private: System::Windows::Forms::TextBox^ textBox6;

private: System::Windows::Forms::TextBox^ textBox7;

private: System::Windows::Forms::TextBox^ textBox8;

private: System::Windows::Forms::ListBox^ listBox1;

private: System::Windows::Forms::ListBox^ listBox2;

private: System::Windows::Forms::ListBox^ listBox3;

private: System::Windows::Forms::TableLayoutPanel^ tableLayoutPanel1;

private: System::ComponentModel::IContainer^ components;

#pragma endregion

#pragma region Windows Form Designer generated code

private:

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

void InitializeComponent(void)

{

this->components = (gcnew System::ComponentModel::Container());

this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());

this->textBox7 = (gcnew System::Windows::Forms::TextBox());

this->textBox6 = (gcnew System::Windows::Forms::TextBox());

this->textBox5 = (gcnew System::Windows::Forms::TextBox());

this->textBox4 = (gcnew System::Windows::Forms::TextBox());

this->textBox3 = (gcnew System::Windows::Forms::TextBox());

this->textBox2 = (gcnew System::Windows::Forms::TextBox());

this->textBox1 = (gcnew System::Windows::Forms::TextBox());

this->label5 = (gcnew System::Windows::Forms::Label());

this->label4 = (gcnew System::Windows::Forms::Label());

this->label3 = (gcnew System::Windows::Forms::Label());

this->label2 = (gcnew System::Windows::Forms::Label());

this->label1 = (gcnew System::Windows::Forms::Label());

this->groupBox2 = (gcnew System::Windows::Forms::GroupBox());

this->label6 = (gcnew System::Windows::Forms::Label());

this->groupBox3 = (gcnew System::Windows::Forms::GroupBox());

this->listBox1 = (gcnew System::Windows::Forms::ListBox());

this->groupBox4 = (gcnew System::Windows::Forms::GroupBox());

this->label8 = (gcnew System::Windows::Forms::Label());

this->listBox3 = (gcnew System::Windows::Forms::ListBox());

this->label7 = (gcnew System::Windows::Forms::Label());

this->listBox2 = (gcnew System::Windows::Forms::ListBox());

this->groupBox5 = (gcnew System::Windows::Forms::GroupBox());

this->tableLayoutPanel1 = (gcnew System::Windows::Forms::TableLayoutPanel());

this->button1 = (gcnew System::Windows::Forms::Button());

this->button2 = (gcnew System::Windows::Forms::Button());

this->timer1 = (gcnew System::Windows::Forms::Timer(this->components));

this->button4 = (gcnew System::Windows::Forms::Button());

this->label9 = (gcnew System::Windows::Forms::Label());

this->textBox8 = (gcnew System::Windows::Forms::TextBox());

this->groupBox1->SuspendLayout();

this->groupBox2->SuspendLayout();

this->groupBox3->SuspendLayout();

this->groupBox4->SuspendLayout();

this->groupBox5->SuspendLayout();

this->SuspendLayout();

//

// groupBox1

//

this->groupBox1->Controls->Add(this->textBox8);

this->groupBox1->Controls->Add(this->label9);

this->groupBox1->Controls->Add(this->textBox7);

this->groupBox1->Controls->Add(this->textBox6);

this->groupBox1->Controls->Add(this->textBox5);

this->groupBox1->Controls->Add(this->textBox4);

this->groupBox1->Controls->Add(this->textBox3);

this->groupBox1->Controls->Add(this->textBox2);

this->groupBox1->Controls->Add(this->textBox1);

this->groupBox1->Controls->Add(this->label5);

this->groupBox1->Controls->Add(this->label4);

this->groupBox1->Controls->Add(this->label3);

this->groupBox1->Controls->Add(this->label2);

this->groupBox1->Controls->Add(this->label1);

this->groupBox1->Location = System::Drawing::Point(1162, 80);

this->groupBox1->Name = L"groupBox1";

this->groupBox1->Size = System::Drawing::Size(404, 376);

this->groupBox1->TabIndex = 0;

this->groupBox1->TabStop = false;

this->groupBox1->Text = L"Настройки";

//

// textBox7

//

this->textBox7->Location = System::Drawing::Point(318, 197);

this->textBox7->Name = L"textBox7";

this->textBox7->Size = System::Drawing::Size(68, 26);

this->textBox7->TabIndex = 11;

this->textBox7->Text = L"20";

this->textBox7->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;

//

// textBox6

//

this->textBox6->Location = System::Drawing::Point(235, 196);

this->textBox6->Name = L"textBox6";

this->textBox6->Size = System::Drawing::Size(68, 26);

this->textBox6->TabIndex = 10;

this->textBox6->Text = L"5";

this->textBox6->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;

//

// textBox5

//

this->textBox5->Location = System::Drawing::Point(235, 164);

this->textBox5->Name = L"textBox5";

this->textBox5->Size = System::Drawing::Size(68, 26);

this->textBox5->TabIndex = 9;

this->textBox5->Text = L"1";

this->textBox5->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;

//

// textBox4

//

this->textBox4->Location = System::Drawing::Point(318, 164);

this->textBox4->Name = L"textBox4";

this->textBox4->Size = System::Drawing::Size(71, 26);

this->textBox4->TabIndex = 8;

this->textBox4->Text = L"20";

this->textBox4->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;

//

// textBox3

//

this->textBox3->Location = System::Drawing::Point(293, 100);

this->textBox3->Name = L"textBox3";

this->textBox3->Size = System::Drawing::Size(96, 26);

this->textBox3->TabIndex = 7;

this->textBox3->Text = L"0.5";

this->textBox3->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;

//

// textBox2

//

this->textBox2->Location = System::Drawing::Point(294, 65);

this->textBox2->Name = L"textBox2";

this->textBox2->Size = System::Drawing::Size(96, 26);

this->textBox2->TabIndex = 6;

this->textBox2->Text = L"1000";

this->textBox2->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;

//

// textBox1

//

this->textBox1->Location = System::Drawing::Point(294, 33);

this->textBox1->Name = L"textBox1";

this->textBox1->Size = System::Drawing::Size(96, 26);

this->textBox1->TabIndex = 5;

this->textBox1->Text = L"20";

this->textBox1->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;

//

// label5

//

this->label5->AutoSize = true;

this->label5->Location = System::Drawing::Point(18, 197);

this->label5->Name = L"label5";

this->label5->Size = System::Drawing::Size(142, 20);

this->label5->TabIndex = 4;

this->label5->Text = L"Требуемые такты";

this->label5->TextAlign = System::Drawing::ContentAlignment::TopCenter;

//

// label4

//

this->label4->AutoSize = true;

this->label4->Location = System::Drawing::Point(18, 164);

this->label4->Name = L"label4";

this->label4->Size = System::Drawing::Size(186, 20);

this->label4->TabIndex = 3;

this->label4->Text = L"Требуемые процессоры";

this->label4->TextAlign = System::Drawing::ContentAlignment::TopCenter;

//

// label3

//

this->label3->AutoSize = true;

this->label3->Location = System::Drawing::Point(18, 100);

this->label3->Name = L"label3";

this->label3->Size = System::Drawing::Size(244, 20);

this->label3->TabIndex = 2;

this->label3->Text = L"Вероятность создания задачи";

this->label3->TextAlign = System::Drawing::ContentAlignment::TopCenter;

//

// label2

//

this->label2->AutoSize = true;

this->label2->Location = System::Drawing::Point(18, 68);

this->label2->Name = L"label2";

this->label2->Size = System::Drawing::Size(203, 20);

this->label2->TabIndex = 1;

this->label2->Text = L"Время выполнения такта";

this->label2->TextAlign = System::Drawing::ContentAlignment::TopCenter;

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(18, 33);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(201, 20);

this->label1->TabIndex = 0;

this->label1->Text = L"Количество процессоров";

this->label1->TextAlign = System::Drawing::ContentAlignment::TopCenter;

//

// groupBox2

//

this->groupBox2->Controls->Add(this->label6);

this->groupBox2->Location = System::Drawing::Point(1162, 462);

this->groupBox2->Name = L"groupBox2";

this->groupBox2->Size = System::Drawing::Size(404, 472);

this->groupBox2->TabIndex = 1;

this->groupBox2->TabStop = false;

this->groupBox2->Text = L"Статистика";

//

// label6

//

this->label6->AutoSize = true;

this->label6->Location = System::Drawing::Point(22, 40);

this->label6->Name = L"label6";

this->label6->Size = System::Drawing::Size(0, 20);

this->label6->TabIndex = 0;

//

// groupBox3

//

this->groupBox3->Controls->Add(this->listBox1);

this->groupBox3->Location = System::Drawing::Point(227, 770);

this->groupBox3->Name = L"groupBox3";

this->groupBox3->Size = System::Drawing::Size(929, 164);

this->groupBox3->TabIndex = 2;

this->groupBox3->TabStop = false;

this->groupBox3->Text = L"Лог";

//

// listBox1

//

this->listBox1->FormattingEnabled = true;

this->listBox1->ItemHeight = 20;

this->listBox1->Location = System::Drawing::Point(6, 22);

this->listBox1->Name = L"listBox1";

this->listBox1->Size = System::Drawing::Size(917, 124);

this->listBox1->TabIndex = 0;

//

// groupBox4

//

this->groupBox4->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>(((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Bottom)

| System::Windows::Forms::AnchorStyles::Left));

this->groupBox4->Controls->Add(this->label8);

this->groupBox4->Controls->Add(this->listBox3);

this->groupBox4->Controls->Add(this->label7);

this->groupBox4->Controls->Add(this->listBox2);

this->groupBox4->Location = System::Drawing::Point(2, 12);

this->groupBox4->Name = L"groupBox4";

this->groupBox4->Size = System::Drawing::Size(219, 1225);

this->groupBox4->TabIndex = 3;

this->groupBox4->TabStop = false;

this->groupBox4->Text = L"Задачи";

//

// label8

//

this->label8->AutoSize = true;

this->label8->Location = System::Drawing::Point(120, 21);

this->label8->Name = L"label8";

this->label8->Size = System::Drawing::Size(84, 20);

this->label8->TabIndex = 3;

this->label8->Text = L"Процессы";

//

// listBox3

//

this->listBox3->FormattingEnabled = true;

this->listBox3->ItemHeight = 20;

this->listBox3->Location = System::Drawing::Point(119, 45);

this->listBox3->Name = L"listBox3";

this->listBox3->Size = System::Drawing::Size(90, 864);

this->listBox3->TabIndex = 2;

//

// label7

//

this->label7->AutoSize = true;

this->label7->Location = System::Drawing::Point(15, 22);

this->label7->Name = L"label7";

this->label7->Size = System::Drawing::Size(77, 20);

this->label7->TabIndex = 1;

this->label7->Text = L"Очередь";

//

// listBox2

//

this->listBox2->FormattingEnabled = true;

this->listBox2->ItemHeight = 20;

this->listBox2->Location = System::Drawing::Point(10, 45);

this->listBox2->Name = L"listBox2";

this->listBox2->Size = System::Drawing::Size(89, 864);

this->listBox2->TabIndex = 0;

//

// groupBox5

//

this->groupBox5->Controls->Add(this->tableLayoutPanel1);

this->groupBox5->Location = System::Drawing::Point(227, 12);

this->groupBox5->Name = L"groupBox5";

this->groupBox5->Size = System::Drawing::Size(929, 752);

this->groupBox5->TabIndex = 4;

this->groupBox5->TabStop = false;

this->groupBox5->Text = L"Процессоры";

//

// tableLayoutPanel1

//

this->tableLayoutPanel1->AutoScroll = true;

this->tableLayoutPanel1->ColumnCount = 5;

this->tableLayoutPanel1->ColumnStyles->Add((gcnew System::Windows::Forms::ColumnStyle(System::Windows::Forms::SizeType::Percent,

20)));

this->tableLayoutPanel1->ColumnStyles->Add((gcnew System::Windows::Forms::ColumnStyle(System::Windows::Forms::SizeType::Percent,

20)));

this->tableLayoutPanel1->ColumnStyles->Add((gcnew System::Windows::Forms::ColumnStyle(System::Windows::Forms::SizeType::Percent,

20)));

this->tableLayoutPanel1->ColumnStyles->Add((gcnew System::Windows::Forms::ColumnStyle(System::Windows::Forms::SizeType::Percent,

20)));

this->tableLayoutPanel1->ColumnStyles->Add((gcnew System::Windows::Forms::ColumnStyle(System::Windows::Forms::SizeType::Percent,

20)));

this->tableLayoutPanel1->Location = System::Drawing::Point(6, 21);

this->tableLayoutPanel1->Name = L"tableLayoutPanel1";

this->tableLayoutPanel1->Size = System::Drawing::Size(917, 721);

this->tableLayoutPanel1->TabIndex = 0;

//

// button1

//

this->button1->Location = System::Drawing::Point(1162, 12);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(131, 62);

this->button1->TabIndex = 5;

this->button1->Text = L"Старт";

this->button1->UseVisualStyleBackColor = true;

this->button1->Click += gcnew System::EventHandler(this, &MyForm::button1\_Click);

//

// button2

//

this->button2->Location = System::Drawing::Point(1299, 12);

this->button2->Name = L"button2";

this->button2->Size = System::Drawing::Size(126, 62);

this->button2->TabIndex = 6;

this->button2->Text = L"Стоп";

this->button2->UseVisualStyleBackColor = true;

this->button2->Click += gcnew System::EventHandler(this, &MyForm::button2\_Click);

//

// timer1

//

this->timer1->Tick += gcnew System::EventHandler(this, &MyForm::timer1\_Tick);

//

// button4

//

this->button4->BackColor = System::Drawing::SystemColors::Control;

this->button4->Location = System::Drawing::Point(1431, 12);

this->button4->Name = L"button4";

this->button4->Size = System::Drawing::Size(112, 62);

this->button4->TabIndex = 7;

this->button4->Text = L"Очистить";

this->button4->UseVisualStyleBackColor = true;

this->button4->Click += gcnew System::EventHandler(this, &MyForm::button4\_Click);

//

// label9

//

this->label9->AutoSize = true;

this->label9->Location = System::Drawing::Point(18, 132);

this->label9->Name = L"label9";

this->label9->Size = System::Drawing::Size(134, 20);

this->label9->TabIndex = 12;

this->label9->Text = L"Размер очереди";

//

// textBox8

//

this->textBox8->Location = System::Drawing::Point(294, 132);

this->textBox8->Name = L"textBox8";

this->textBox8->Size = System::Drawing::Size(96, 26);

this->textBox8->TabIndex = 13;

this->textBox8->Text = L"100";

this->textBox8->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(9, 20);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(1578, 946);

this->Controls->Add(this->button4);

this->Controls->Add(this->button2);

this->Controls->Add(this->button1);

this->Controls->Add(this->groupBox3);

this->Controls->Add(this->groupBox5);

this->Controls->Add(this->groupBox4);

this->Controls->Add(this->groupBox2);

this->Controls->Add(this->groupBox1);

this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::FixedSingle;

this->Name = L"MyForm";

this->Text = L"Эмулятор работы многопроцессорной системы";

this->groupBox1->ResumeLayout(false);

this->groupBox1->PerformLayout();

this->groupBox2->ResumeLayout(false);

this->groupBox2->PerformLayout();

this->groupBox3->ResumeLayout(false);

this->groupBox4->ResumeLayout(false);

this->groupBox4->PerformLayout();

this->groupBox5->ResumeLayout(false);

this->ResumeLayout(false);

}

#pragma endregion

#pragma region Service Functions

private: void CenterGroupBoxTitle(GroupBox^ groupbox)

{

Label^ label = gcnew Label();

label->Text = groupbox->Text;

groupbox->Text = "";

label->Left = groupbox->Left + (groupbox->Width - label->Width) / 2;

label->Top = groupbox->Top;

label->Parent = groupbox->Parent;

label->BringToFront();

this->Controls->Add(label);

}

private: Task popQueue() {

Task tmp = this->taskQueue->Get();

this->listBox2->Items->Remove(gcnew String(std::format("{}:{},{}", tmp.id, tmp.procCount, tmp.tactCount).c\_str()));

return tmp;

}

private: void PushQueue(const Task& task) {

this->taskQueue->Push(task);

this->listBox2->Items->Add(gcnew String(std::format("{}:{},{}", task.id, task.procCount, task.tactCount).c\_str()));

}

private: void tryCreateNewTask() {

// Создание задачи (по вероятности) id:кол-во процессоров, кол-во тактов

if ((double)rand() / (RAND\_MAX) <= this->taskProbability) {

Task task;

task.id = ++this->statistic->taskCount;

task.procCount = minRequiredProcessor + rand() % (maxRequiredProcessor - minRequiredProcessor + 1);

task.tactCount = minRequiredTact + rand() % (maxRequiredTact - minRequiredTact + 1);

task.color = task.id % this->colors->GetLength(0);

PushQueue(task);

this->listBox1->Items->Add(gcnew String(

std::format("Задача {}, состоящая из {} тактов, требующая {} процессоров, постуает в очередь", task.id, task.tactCount, task.procCount).c\_str())

);

this->listBox1->TopIndex = this->listBox1->Items->Count - 1;

}

}

private: void updateStatistic()

{

std::string formated =

std::format("Выполнено тактов: {}\n", this->statistic->tactCount) +

std::format("Среденее кол-во занятых процессоров: {:.2f}\n", this->statistic->middleBusyProcessor) +

std::format("Общее кол-во тактов бездействия: {}\n", this->statistic->tactInactionCount) +

std::format("Кол-во выполняемых задач: {}\n", this->statistic->taskInWorkCount) +

std::format("Кол-во задач в очереди: {}\n", this->taskQueue->GetCount()) +

std::format("Выполнено задач: {}\n", this->statistic->taskComplitedCount) +

std::format("Всего задач поступило: {}", this->statistic->taskCount);

label6->Text = gcnew String(formated.c\_str());

}

private: void updateProcessorGraphics() {

for (size\_t i = 0; i < std::ceil(this->processorCount / (double)TABLE\_COLUMS); i++) {

for (size\_t j = 0; j < TABLE\_COLUMS; j++) {

if (i \* TABLE\_COLUMS + j >= processorCount)

break;

Processor proc = this->processors->at(i \* TABLE\_COLUMS + j);

auto element = this->tableLayoutPanel1->GetControlFromPosition(j, i);

if (proc.isEmpty) {

element->Text = "";

element->BackColor = Color::LightGray;

}

else {

element->Text = gcnew String(std::format("{}:{}", proc.task.id, proc.taskTactCount).c\_str());

element->BackColor = this->colors[proc.task.color];

}

}

}

}

private: void updateDiagram() {

Graphics^ gr = this->groupBox2->CreateGraphics();

gr->Clear(SystemColors::Control);

if (!this->taskQueue->IsEmpty()) {

Pen^ bp = gcnew Pen(Color::Black);

bp->Width = 10.0f;

auto startAngle = this->taskQueue->GetInd() \* 360.0 / this->taskQueue->GetSize();

auto endAngle = (this->taskQueue->GetInd() + this->taskQueue->GetCount()) \* 360.0 / this->taskQueue->GetSize();

gr->DrawArc(bp, 60, 135, 150, 150, startAngle, endAngle - startAngle);

}

}

#pragma endregion

#pragma region Event Hendlers

private: System::Void timer1\_Tick(System::Object^ sender, System::EventArgs^ e) {

size\_t freeProcessors = 0;

std::set<int> complited;

for (auto proc = this->processors->begin(); proc < this->processors->end(); proc++) {

proc->stepCount++;

if (proc->isEmpty) {

proc->emptyTactCount++;

this->statistic->tactInactionCount++;

}

else {

proc->taskTactCount--;

if (proc->taskTactCount == 0) {

complited.insert(proc->task.id);

proc->isEmpty = true;

listBox3->Items->Remove(gcnew String(

std::format("{}:{},{}", proc->task.id, proc->task.procCount, proc->task.tactCount).c\_str()

));

}

}

if (proc->isEmpty)

freeProcessors++;

}

this->statistic->middleBusyProcessor = (this->statistic->middleBusyProcessor \* this->statistic->tactCount +

(this->processorCount - freeProcessors)) / (this->statistic->tactCount + 1);

this->statistic->taskComplitedCount += complited.size();

this->statistic->taskInWorkCount -= complited.size();

if (!this->taskQueue->IsFull())

tryCreateNewTask();

if (!this->taskQueue->IsEmpty()) {

Task task = popQueue();

std::string status;

if (task.procCount <= freeProcessors) {

status = std::format("Задача {}, требующая {} процессоров, поступает на процессоры",

task.id, task.procCount);

auto listBoxItem = gcnew String(

std::format("{}:{},{}", task.id, task.procCount, task.tactCount).c\_str()

);

this->listBox3->Items->Add(listBoxItem);

this->statistic->taskInWorkCount++;

for (auto proc = this->processors->begin(); proc < this->processors->end(); proc++) {

if (task.procCount == 0)

break;

if (proc->isEmpty) {

proc->task = task;

proc->taskTactCount = task.tactCount;

proc->isEmpty = false;

task.procCount--;

}

}

}

else {

PushQueue(task);

status = std::format("Задачу {}, требующую {} процессоров, невозможно выполнить, в данный момент свободных процессоров {}",

task.id, task.procCount, freeProcessors);

}

this->listBox1->Items->Add(gcnew String(status.c\_str()));

this->listBox1->TopIndex = this->listBox1->Items->Count - 1;

}

this->statistic->tactCount++;

updateProcessorGraphics();

updateDiagram();

updateStatistic();

}

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {

this->timer1->Enabled = false;

}

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

if (this->textBox1->ReadOnly == true) {

this->timer1->Enabled = true;

return;

}

try {

this->processorCount = Int32::Parse(textBox1->Text);

this->tactTime = Int32::Parse(textBox2->Text);

this->taskProbability = Double::Parse(textBox3->Text);

this->minRequiredProcessor = Int32::Parse(textBox5->Text);

this->maxRequiredProcessor = Int32::Parse(textBox4->Text);

this->minRequiredTact = Int32::Parse(textBox6->Text);

this->maxRequiredTact = Int32::Parse(textBox7->Text);

this->queueSize = Int32::Parse(textBox8->Text);

this->taskQueue = new TQueue<Task>(queueSize);

if (this->minRequiredProcessor > this->maxRequiredProcessor ||

this->minRequiredTact > this->maxRequiredTact)

{

throw gcnew FormatException;

}

{

this->textBox1->ReadOnly = true;

this->textBox2->ReadOnly = true;

this->textBox3->ReadOnly = true;

this->textBox4->ReadOnly = true;

this->textBox5->ReadOnly = true;

this->textBox6->ReadOnly = true;

this->textBox7->ReadOnly = true;

this->textBox8->ReadOnly = true;

}

for (size\_t i = 0; i < std::ceil(this->processorCount / (double)TABLE\_COLUMS); i++) {

for (size\_t j = 0; j < TABLE\_COLUMS; j++) {

if (i \* TABLE\_COLUMS + j >= processorCount)

break;

processors->push\_back(Processor());

Button^ procButton = gcnew Button();

procButton->BackColor = Color::LightGray;

procButton->Size = System::Drawing::Size(100, 100);

procButton->ForeColor = Color::Black;

procButton->Font = gcnew System::Drawing::Font("Microsoft Sans Serif", 10);

procButton->Text = "";

this->tableLayoutPanel1->Controls->Add(procButton, j, i);

}

}

this->timer1->Interval = this->tactTime;

this->timer1->Enabled = true;

}

catch (FormatException^ e) {

MessageBox::Show("Ошибка ввода");

}

}

private: System::Void button4\_Click(System::Object^ sender, System::EventArgs^ e) {

this->timer1->Enabled = false;

{

this->textBox1->ReadOnly = false;

this->textBox2->ReadOnly = false;

this->textBox3->ReadOnly = false;

this->textBox4->ReadOnly = false;

this->textBox5->ReadOnly = false;

this->textBox6->ReadOnly = false;

this->textBox7->ReadOnly = false;

this->textBox8->ReadOnly = false;

}

this->listBox1->Items->Clear();

this->listBox2->Items->Clear();

this->listBox3->Items->Clear();

this->tableLayoutPanel1->Controls->Clear();

this->groupBox2->CreateGraphics()->Clear(SystemColors::Control);

this->statistic = new Statistic();

this->processors = new std::vector<Processor>;

this->taskQueue = new TQueue<Task>(queueSize);

updateStatistic();

}

#pragma endregion

};

}

1. Приложение 8

ServiceTypes.h

#ifndef \_QUEUE\_GUI\_SERVICE\_TYPES\_

#define \_QUEUE\_GUI\_SERVICE\_TYPES\_

#include <cstdint>

struct Statistic {

uint64\_t tactCount = 0;

double middleBusyProcessor = 0;

uint64\_t tactInactionCount = 0;

uint64\_t taskComplitedCount = 0;

uint64\_t taskInWorkCount = 0;

uint64\_t taskSkipedCount = 0;

uint64\_t taskCount = 0;

};

struct Task {

int id;

size\_t tactCount;

size\_t procCount;

size\_t color;

};

struct Processor {

Task task;

size\_t stepCount = 0;

size\_t taskTactCount = 0;

size\_t emptyTactCount = 0;

bool isEmpty = true;

};

#endif \_QUEUE\_GUI\_SERVICE\_TYPES\_