Министерство образования и науки Российской Федерации

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**Разработка структуры данных очередь и её применение для решения практической задачи**

**Выполнил**:студент группы 381606-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тимакин Н.Е.

Подпись

**Проверил**: к.ф.-м.н., доц.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Баркалов К.А.

Подпись

Нижний Новгород 2017

**Введение**

Очередь – это структура данных, представляющая собой список элементов и организованная по принципу: «первый пришёл – первый ушёл», то есть добавление элемента происходит только в конец, а удаление – только из начала. Очередь не только часто встречается в повседневной жизни (в магазинах, на почте, в банках), но ещё с её помощью решаются многие прикладные задачи. Одна из таких задач и будет рассмотрена как пример использования этой структуры данных.

**Постановка задачи**

Реализовать структуру данных очередь как шаблонный класс, написать к нему тесты, подтверждающие его работоспособность, и тестовую программу – имитацию работы многопроцессорной системы в виде приложения с графическим интерфейсом.

**Описание структуры программы**

Программа содержит в себе 3 проекта:

1. Queue lab – реализация очереди
   * TQueue.h – объявление и реализация класса TQueue
   * main.cpp – файл с функцией main()
2. Test – тесты для очереди
   * gtest.h – заголовочный файл библиотеки Google Test
   * gtest-all.cc – реализация библиотеки Google Test
   * test\_main.cpp – файл, запускающий все тесты
   * test\_queue.cpp – тесты для класса TQueue
3. ClusterImmitation – приложение с графическим интерфейсом
   * Structs.h – файл с объявлением структур для реализации алгоритма
   * MyForm.h – объявление и реализация формы для графического приложения
   * MyForm.cpp – файл для запуска приложения

T – шаблон

**Класс TQueue**

Поля:

int first – номер первого элемента

int last – номер следующего за последним элемента

int size – текущий размер очереди

int maxsize – максимальный размер очереди

T\* ring – память для хранения элементов

Методы:

TQueue(int \_maxsize = 10) – конструктор по умолчанию

TQueue(const TQueue& q) – конструктор копирования

~TQueue() - деструктор

int isempty() – проверка на пустоту

int isfull() – проверка на полноту

void push(const T elem) – добавить элемент

T pop() – удалить и получить элемент

T getLastElem() – получить последний элемент

int getFirst() – получить номер первого элемента

int getSize() – получить текущий размер

int getMaxsize() – получить максимальный размер

T top() – получить первый элемент

int getLast() – получить номер следующего за последним элемента

int operator==(const TQueue& q) const – перегрузка операции сравнения

int operator!=(const TQueue& q) const – перегрузка операции не равно

**Описание алгоритмов**

**Класс TQueue**

Главная идея реализации очереди состоит в её представлении кольцевым буфером, чтобы не перепаковывать массив элементов на каждой итерации программы. Для этого необходимо индексировать массив таким образом, чтобы следующим элементом для элемента с номером maxsize-1 был нулевой. Это легко достигается, если обращаться к элементу по остатку от деления номера на максимальный размер (тогда элемент с номером maxsize станет нулевым, maxsize+1 – первым и т.д.)

TQueue(int \_maxsize = 10) {если \_maxsize <= 0, выбросить исключение. Максимальному размеру присвоить \_maxsize. Выделить память объёмом максимального размера. Номеру первого элемента присвоить 0, последнего - -1. Текущему размеру присвоить 0}

TQueue(const TQueue& q) {максимальному размеру присвоить максимальный размер q. Выделить память объёмом максимального размера. Номеру первого элемента присвоить номер первого элемента q, последнего – номер последнего q. Текущему размеру присвоить текущий размер q. Присвоить каждому элементу соответствующий из q}

~TQueue() {освободить память}

int isempty() {если текущий размер 0, то вернуть 1, иначе – 0}

int isfull() {если текущий размер равен максимальному, вернуть 1, иначе – 0}

void push(const T elem) {если очередь полная, то выбросить исключение. Последнему присвоить остаток от деления последнего+1 на максимальный размер. Последнему элементу присвоить значение elem. Увеличить размер на 1}

T pop() {если очередь пустая, выбросить исключение. Запомнить номер первого элемента во временной переменной. Первому присвоить остаток от деления первого на максимальный размер. Уменьшить размер на 1. Вернуть элемент с номером, который запомнили ранее}

T getLastElem() {вернуть элемент с последним номером}

int getFirst() {вернуть номер первого элемента}

int getSize() {вернуть текущий размер}

int getMaxsize() {вернуть максимальный размер}

T top() {если очередь пустая, выбросить исключение. Вернуть элемент с первым номером}

int getLast() {вернуть номер следующего за последним элемента}

int operator==(const TQueue& q) const {если максимальные размеры, текущие размеры, первые номера или последние номера не равны, то вернуть 0. Если хотя бы один из элементов с первого до последнего не равен, вернуть 0. Вернуть 1}

int operator!=(const TQueue& q) const {вернуть отрицание от проверки на равенство \*this и q}

**Приложение с графическим интерфейсом**

Для реализации имитации работы многопроцессорной системы нам понадобятся две структуры: задание (описывает характеристики задачи, которую будет обрабатывать система) и процессор (описывает каждый конкретный узел системы). Эти две структуры и описаны в файле Structs.h

Поля задания:

1. Количество тактов, требуемое для завершения задачи
2. Количество процессоров, требуемое для завершения задачи
3. Номер задачи
4. Цвет задачи

Поля процессора:

1. Индикатор занятости
2. Задание, обрабатываемое в данный момент
3. Оставшееся количество тактов до завершения задачи
4. Общее количество тактов текущей задачи
5. Количество тактов простоя
6. Координата по X
7. Координата по Y

Для работы системы инициализируются несколько объектов как поля формы: объект типа Graphics, очередь для задач и массив процессоров – это 3 структуры, на которые опирается вся программа.

До начала работы программы необходимо ввести в соответствующе поля на форме количество процессоров, вероятность возникновения новой задачи, максимальное количество узлов, которое может занять одна задача, и максимальное количество тактов, которое может выполняться одна задача.

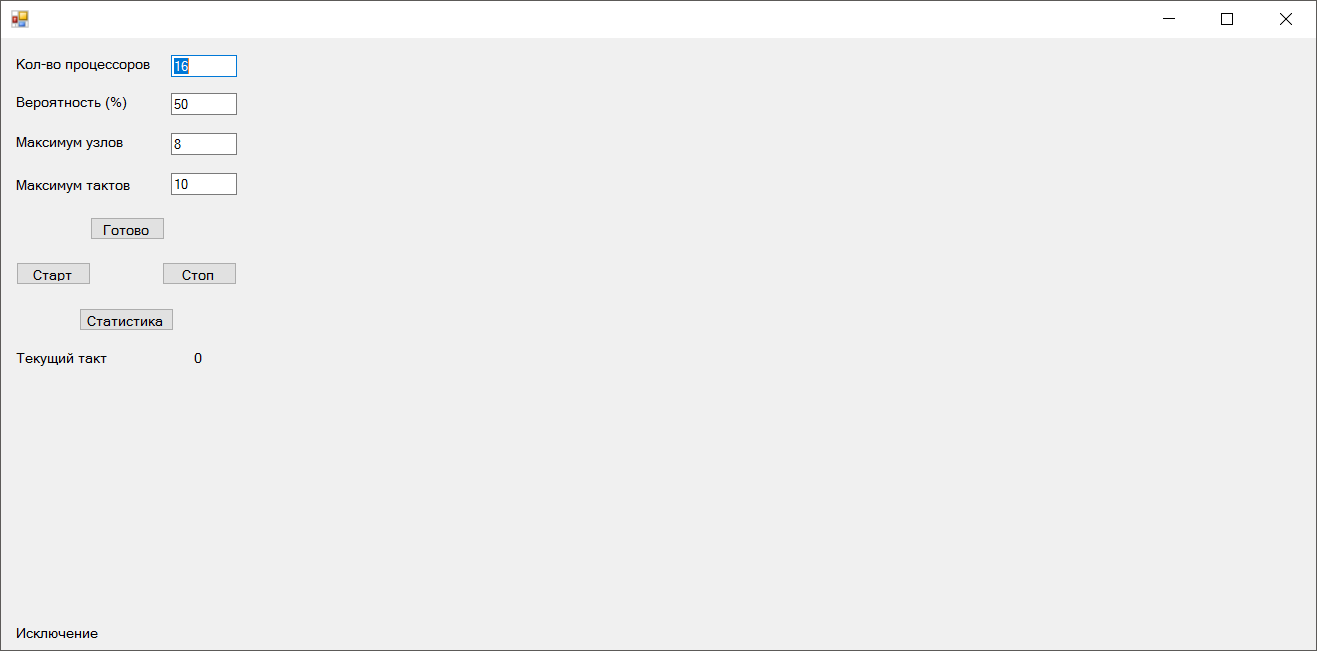
После того как все поля проинициализированы, по нажатию кнопки «готово» происходит выделение памяти под задачи и процессоры, все переменные приводятся в готовность для начала работы, отрисовываются ячейки под нужное количество процессоров и задач.

По нажатию кнопки «старт» включается таймер. На каждом такте, если случайно генерируемое число больше вероятности и очередь не полная, то создаётся новая задача со случайными характеристиками. Затем обновляется визуализация очереди. После этого пока есть свободные процессоры для следующей задачи, эта задача их занимает, обновляется визуализация процессоров. Пробегаются все процессоры, и в зависимости от их занятости изменяются счётчики для статистики, процессоры освобождаются, и обновляется визуализация. По нажатию кнопки «стоп» таймер выключается.

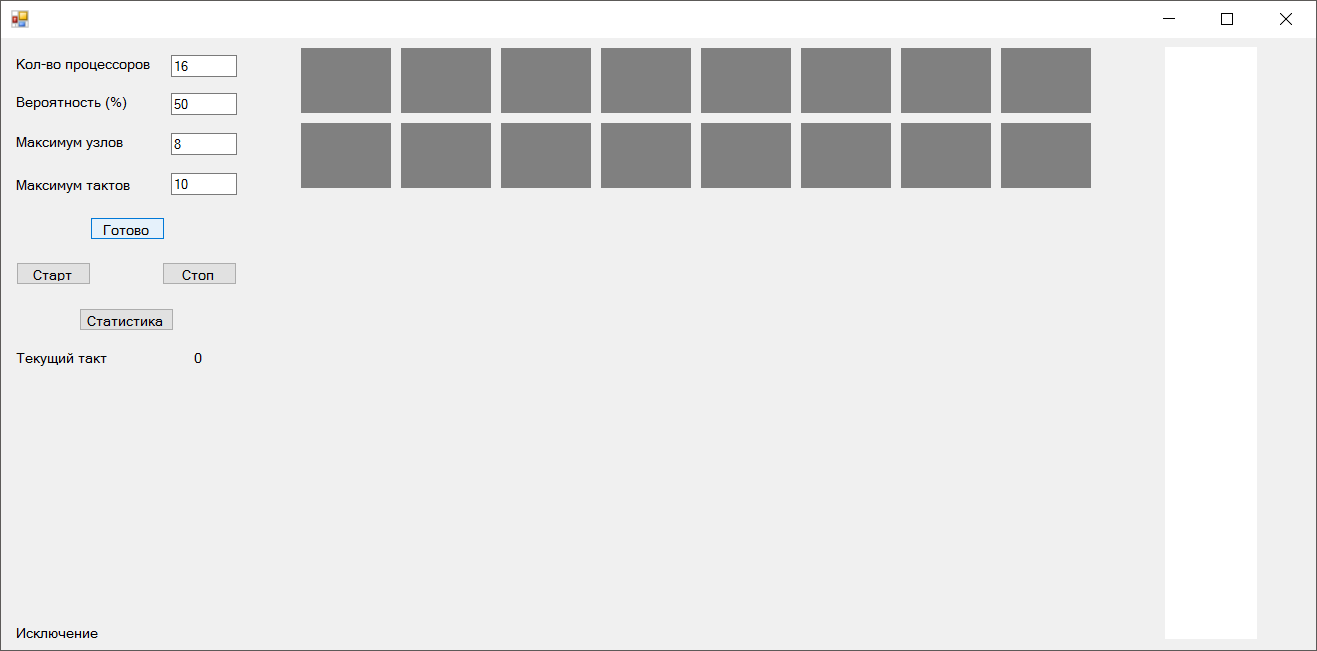
По нажатию кнопки «статистика» возникает MessegeBox, в котором приведена вся статистика.

**Результаты**

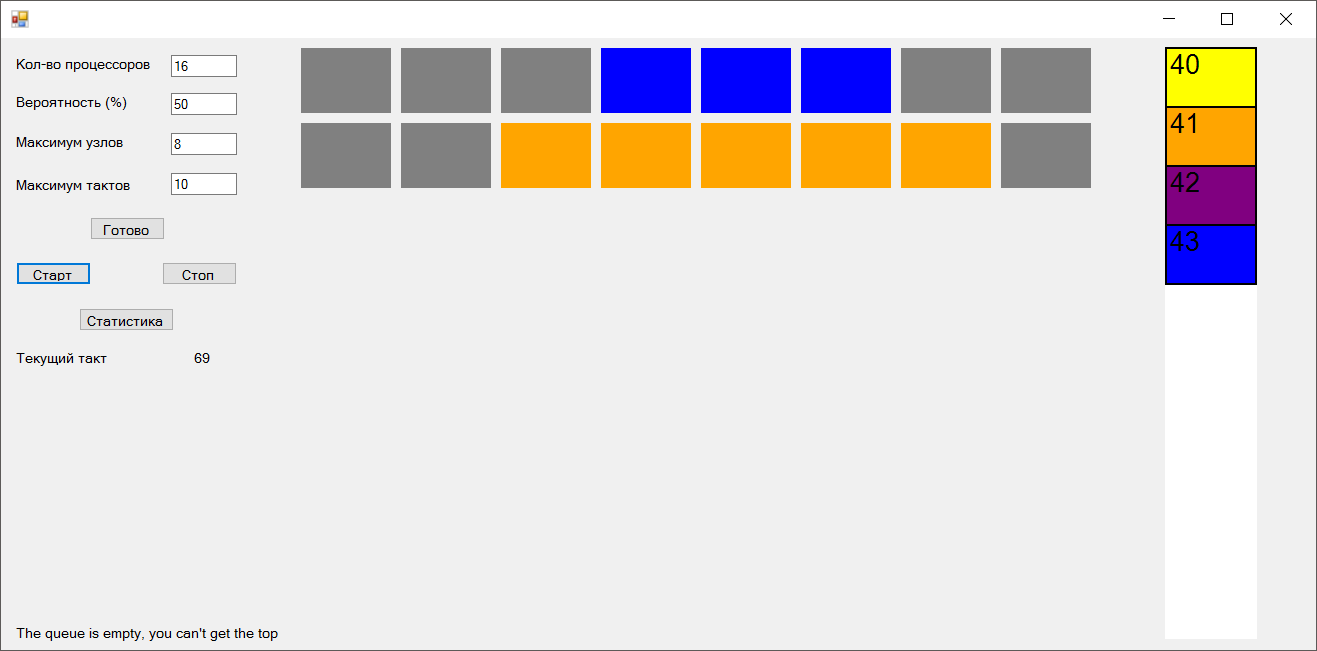
Начальный этап



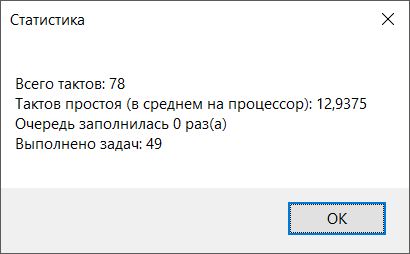
Подготовительный этап



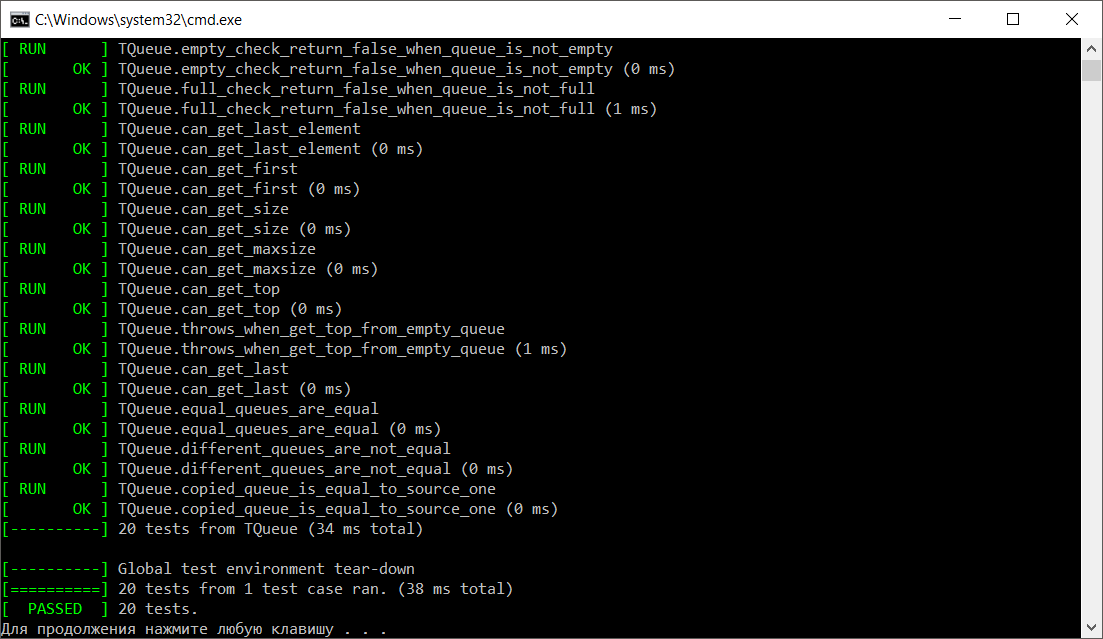
Во время работы



Окно статистики



Тесты



**Вывод**

Тестовая программа для имитации работы многопроцессорной системы работает корректно, все тесты пройдены, все задачи, поставленные ранее, выполнены, а значит, разработку структуры данных очередь и её применение в практической задаче можно считать успешными.

**Литература**

Брайан Керниган, Деннис Ритчи «Язык программирования Си»

Брюс Эккель «Философия С++. Введение в стандартный С++»

Стивен Прата «Язык программирования С++. Лекции и упражнения»

**Приложение**

**TQueue.h**

#pragma once

#include <iostream>

#include <stdlib.h>

using namespace std;

template <class T>

class TQueue {

int first;

int last;

int size;

int maxsize;

T\* ring;

public:

TQueue(int \_maxsize = 10) {

if (\_maxsize <= 0) {

char \*str = "Queue's maxsize must be a positive number";

throw str;

}

maxsize = \_maxsize;

ring = new T[maxsize];

first = 0;

last = -1;

size = 0;

}

TQueue(const TQueue& q) {

maxsize = q.maxsize;

ring = new T[maxsize];

first = q.first;

last = q.last;

size = q.size;

for (int i = 0; i < maxsize; i++) {

ring[i] = q.ring[i];

}

}

~TQueue() {

delete[] ring;

}

int isempty() {

if (size == 0)

return 1;

return 0;

}

int isfull() {

if (size == maxsize)

return 1;

return 0;

}

void push(const T elem) {

if (isfull()) {

char \*str = "The queue is full, you can't add a new element";

throw str;

}

last = (last + 1) % maxsize;

ring[last] = elem;

size++;

}

T pop() {

if (isempty()) {

char \*str = "The queue is empty, you can't delete an element";

throw str;

}

int tmp = first;

first = (first + 1) % maxsize;

size--;

return ring[tmp];

}

T getLastElem() {

return ring[last];

}

int getFirst() {

return first;

}

int getSize() {

return size;

}

int getMaxsize() {

return maxsize;

}

T top() {

if (isempty()) {

char \*str = "The queue is empty, you can't get the top";

throw str;

}

return ring[first];

}

int getLast(){

return last;

}

int operator==(const TQueue& q) const {

if ((maxsize != q.maxsize) || (size != q.size) || (first != q.first) || (last != q.last))

return 0;

for (int i = first; i < last; i++) {

if (ring[i] != q.ring[i])

return 0;

}

return 1;

}

int operator!=(const TQueue& q) const {

return !(\*this == q);

}

};

**test\_queue.cpp**

#include "..\Queue lab\TQueue.h"

#include "gtest.h"

TEST(TQueue, can\_create\_a\_queue\_with\_positive\_length) {

ASSERT\_NO\_THROW(TQueue<int> q(3));

}

TEST(TQueue, throws\_when\_create\_a\_queue\_with\_negative\_length) {

ASSERT\_ANY\_THROW(TQueue<int> q(-3));

}

TEST(TQueue, can\_push\_element\_in\_not\_empty\_queue) {

TQueue<int> q(3);

ASSERT\_NO\_THROW(q.push(1));

}

TEST(TQueue, throws\_when\_push\_element\_in\_full\_queue) {

TQueue<int> q(3);

for (int i = 0; i < 3; i++) {

q.push(i);

}

ASSERT\_ANY\_THROW(q.push(1));

}

TEST(TQueue, can\_pop\_from\_not\_empty\_queue) {

TQueue<int> q(3);

for (int i = 0; i < 2; i++) {

q.push(i);

}

ASSERT\_NO\_THROW(q.pop());

}

TEST(TQueue, throws\_when\_pop\_from\_empty\_queue) {

TQueue<int> q(3);

ASSERT\_ANY\_THROW(q.pop());

}

TEST(TQueue, empty\_check\_return\_true\_when\_queue\_is\_empty) {

TQueue<int> q(3);

EXPECT\_EQ(1, q.isempty());

}

TEST(TQueue, full\_check\_return\_true\_when\_queue\_is\_full) {

TQueue<int> q(3);

for (int i = 0; i < 3; i++) {

q.push(i);

}

EXPECT\_EQ(1, q.isfull());

}

TEST(TQueue, empty\_check\_return\_false\_when\_queue\_is\_not\_empty) {

TQueue<int> q(3);

q.push(1);

EXPECT\_EQ(0, q.isempty());

}

TEST(TQueue, full\_check\_return\_false\_when\_queue\_is\_not\_full) {

TQueue<int> q(3);

for (int i = 0; i < 2; i++) {

q.push(i);

}

EXPECT\_EQ(0, q.isfull());

}

TEST(TQueue, can\_get\_last\_element) {

TQueue<int> q(3);

for (int i = 0; i < 3; i++) {

q.push(i);

}

EXPECT\_EQ(2, q.getLastElem());

}

TEST(TQueue, can\_get\_first) {

TQueue<int> q(3);

for (int i = 0; i < 3; i++) {

q.push(i);

}

q.pop();

EXPECT\_EQ(1, q.getFirst());

}

TEST(TQueue, can\_get\_size) {

TQueue<int> q(3);

for (int i = 0; i < 2; i++) {

q.push(i);

}

EXPECT\_EQ(2, q.getSize());

}

TEST(TQueue, can\_get\_maxsize) {

TQueue<int> q(5);

EXPECT\_EQ(5, q.getMaxsize());

}

TEST(TQueue, can\_get\_top) {

TQueue<int> q(3);

for (int i = 0; i < 3; i++) {

q.push(i);

}

q.pop();

EXPECT\_EQ(1,q.top());

}

TEST(TQueue, throws\_when\_get\_top\_from\_empty\_queue) {

TQueue<int> q(3);

ASSERT\_ANY\_THROW(q.top());

}

TEST(TQueue, can\_get\_last) {

TQueue<int> q(3);

for (int i = 0; i < 3; i++) {

q.push(i);

}

EXPECT\_EQ(2, q.getLast());

}

TEST(TQueue, equal\_queues\_are\_equal) {

TQueue<int> q(3), p(3);

for (int i = 0; i < 2; i++) {

q.push(i);

p.push(i);

}

EXPECT\_EQ(q, p);

}

TEST(TQueue, different\_queues\_are\_not\_equal) {

TQueue<int> q(3), p(3);

for (int i = 0; i < 2; i++) {

q.push(i);

p.push(i+1);

}

EXPECT\_NE(q, p);

}

TEST(TQueue, copied\_queue\_is\_equal\_to\_source\_one) {

TQueue<int> q(3);

for (int i = 0; i < 2; i++) {

q.push(i);

}

TQueue<int> p(q);

EXPECT\_EQ(q, p);

}

**Structs.h**

#pragma once

struct Task {

int StepCount;

int ProcCount;

int ID;

int Color;

};

struct Processor {

bool free;

int ActiveTaskID;

int CurrentStepCount;

int MaxStepCount;

int InactiveTactsCount;

int id;

int X;

int Y;

};

**MyForm.h**

#pragma once

#include "Structs.h"

#include "..\Queue lab\TQueue.h"

#include <stdlib.h>

int FreeProcCount(Processor \*arr, int \_procCount) {

int res = 0;

for (int i = 0; i < \_procCount; i++) {

if (arr[i].free) {

res++;

}

}

return res;

}

namespace ClusterImmitation {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Summary for MyForm

/// </summary>

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

{

private:

Graphics^ gr;

TQueue<Task> \*qTask;

int TactsCount;

int FullQTaskCount;

Processor \*ProcArray;

int CountOfInactiveTacts;

double InactiveTactsPerProcessor;

int IDCounter;

int CountOfCurrentTasks;

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

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

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

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

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

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

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

public:

MyForm(void)

{

InitializeComponent();

//

//TODO: Add the constructor code here

//

gr = CreateGraphics();

qTask = new TQueue<Task>(10);

}

protected:

/// <summary>

/// Clean up any resources being used.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

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

protected:

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

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

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

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

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

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

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

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

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

private:

/// <summary>

/// Required designer variable.

/// </summary>

#pragma region Windows Form Designer generated code

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

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

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

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

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

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

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

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

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

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

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

this->button3 = (gcnew System::Windows::Forms::Button());

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

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

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

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

this->SuspendLayout();

//

// textBox1

//

this->textBox1->Location = System::Drawing::Point(170, 17);

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

this->textBox1->Size = System::Drawing::Size(66, 22);

this->textBox1->TabIndex = 0;

this->textBox1->Text = L"16";

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(12, 17);

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

this->label1->Size = System::Drawing::Size(142, 17);

this->label1->TabIndex = 1;

this->label1->Text = L"Кол-во процессоров";

//

// label2

//

this->label2->AutoSize = true;

this->label2->Location = System::Drawing::Point(12, 55);

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

this->label2->Size = System::Drawing::Size(119, 17);

this->label2->TabIndex = 3;

this->label2->Text = L"Вероятность (%)";

//

// textBox2

//

this->textBox2->Location = System::Drawing::Point(170, 55);

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

this->textBox2->Size = System::Drawing::Size(66, 22);

this->textBox2->TabIndex = 2;

this->textBox2->Text = L"50";

//

// label3

//

this->label3->AutoSize = true;

this->label3->Location = System::Drawing::Point(12, 138);

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

this->label3->Size = System::Drawing::Size(122, 17);

this->label3->TabIndex = 5;

this->label3->Text = L"Максимум тактов";

//

// textBox3

//

this->textBox3->Location = System::Drawing::Point(170, 135);

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

this->textBox3->Size = System::Drawing::Size(66, 22);

this->textBox3->TabIndex = 4;

this->textBox3->Text = L"10";

//

// label4

//

this->label4->AutoSize = true;

this->label4->Location = System::Drawing::Point(12, 95);

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

this->label4->Size = System::Drawing::Size(115, 17);

this->label4->TabIndex = 7;

this->label4->Text = L"Максимум узлов";

//

// textBox4

//

this->textBox4->Location = System::Drawing::Point(170, 95);

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

this->textBox4->Size = System::Drawing::Size(66, 22);

this->textBox4->TabIndex = 6;

this->textBox4->Text = L"8";

//

// timer1

//

this->timer1->Interval = 1000;

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

//

// button1

//

this->button1->Location = System::Drawing::Point(89, 179);

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

this->button1->Size = System::Drawing::Size(75, 23);

this->button1->TabIndex = 8;

this->button1->Text = L"Готово";

this->button1->UseVisualStyleBackColor = true;

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

//

// button2

//

this->button2->Location = System::Drawing::Point(15, 224);

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

this->button2->Size = System::Drawing::Size(75, 23);

this->button2->TabIndex = 9;

this->button2->Text = L"Старт";

this->button2->UseVisualStyleBackColor = true;

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

//

// button3

//

this->button3->Location = System::Drawing::Point(161, 224);

this->button3->Name = L"button3";

this->button3->Size = System::Drawing::Size(75, 23);

this->button3->TabIndex = 10;

this->button3->Text = L"Стоп";

this->button3->UseVisualStyleBackColor = true;

this->button3->Click += gcnew System::EventHandler(this, &MyForm::button3\_Click);

//

// label5

//

this->label5->AutoSize = true;

this->label5->Location = System::Drawing::Point(12, 311);

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

this->label5->Size = System::Drawing::Size(99, 17);

this->label5->TabIndex = 12;

this->label5->Text = L"Текущий такт";

//

// label6

//

this->label6->AutoSize = true;

this->label6->Location = System::Drawing::Point(12, 586);

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

this->label6->Size = System::Drawing::Size(90, 17);

this->label6->TabIndex = 13;

this->label6->Text = L"Исключение";

//

// button4

//

this->button4->Location = System::Drawing::Point(78, 270);

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

this->button4->Size = System::Drawing::Size(95, 23);

this->button4->TabIndex = 14;

this->button4->Text = L"Статистика";

this->button4->UseVisualStyleBackColor = true;

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

//

// label7

//

this->label7->AutoSize = true;

this->label7->Location = System::Drawing::Point(190, 311);

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

this->label7->Size = System::Drawing::Size(16, 17);

this->label7->TabIndex = 15;

this->label7->Text = L"0";

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

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

this->ClientSize = System::Drawing::Size(1315, 612);

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

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

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

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

this->Controls->Add(this->button3);

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

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

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

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

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

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

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

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

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

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

this->Name = L"MyForm";

this->Text = L" ";

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

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

int ProcCount = Convert::ToInt32(textBox1->Text);

TactsCount = 0;

FullQTaskCount = 0;

CountOfInactiveTacts = 0;

InactiveTactsPerProcessor = 0.0;

IDCounter = 0;

CountOfCurrentTasks = 0;

ProcArray = new Processor[ProcCount];

for (int i = 0; i < ProcCount; i++) {

ProcArray[i].free = true;

ProcArray[i].ActiveTaskID = -1;

ProcArray[i].CurrentStepCount = 0;

ProcArray[i].MaxStepCount = 0;

ProcArray[i].InactiveTactsCount = 0;

ProcArray[i].id = i;

}

int x = 300, y = 10, width = 90, height = 65;

int dx = 100, dy = 75;

for (int i = 0; i < ProcCount; i++) {

ProcArray[i].X = x;

ProcArray[i].Y = y;

gr->FillRectangle(Brushes::Gray, x, y, width, height);

if (i % 8 == 7) {

y += dy;

x = 300;

}

else {

x += dx;

}

}

gr->FillRectangle(Brushes::White, 1164, 9, 92, 592);

}

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

timer1->Enabled = true;

}

private: System::Void button3\_Click(System::Object^ sender, System::EventArgs^ e) {

timer1->Enabled = false;

}

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

try {

int CountOfProc = Convert::ToInt32(textBox1->Text);

int probability = Convert::ToInt32(textBox2->Text);

int MaxTaskProc = Convert::ToInt32(textBox4->Text);

int MaxTaskTact = Convert::ToInt32(textBox3->Text);

float TaskHeight = 590.0 / (qTask->getMaxsize());

Pen ^blackpen = gcnew Pen(Brushes::Black, 2.0f);

if (qTask->isfull()) {

FullQTaskCount++;

}

int chance = rand() % 101;

if ((chance >= (100 - probability)) && (!(qTask->isfull()))) {

Task temp;

temp.ID = IDCounter;

IDCounter++;

temp.ProcCount = rand() % (MaxTaskProc - 1) + 1;

temp.StepCount = rand() % (MaxTaskTact - 1) + 1;

temp.Color = rand() % 7;

qTask->push(temp);

}

if (!(qTask->isempty())) {

gr->FillRectangle(Brushes::White, 1164, 9, 92, 592);

TQueue<Task> CopyQ(\*qTask);

for (int i = 0; i < qTask->getSize(); i++) {

switch (CopyQ.top().Color) {

case 0: gr->FillRectangle(Brushes::Red, 1165.0, (float)10 + TaskHeight \* i, 90.0, TaskHeight); break;

case 1: gr->FillRectangle(Brushes::Orange, 1165.0, (float)10 + TaskHeight \* i, 90.0, TaskHeight); break;

case 2: gr->FillRectangle(Brushes::Yellow, 1165.0, (float)10 + TaskHeight \* i, 90.0, TaskHeight); break;

case 3: gr->FillRectangle(Brushes::Green, 1165.0, (float)10 + TaskHeight \* i, 90.0, TaskHeight); break;

case 4: gr->FillRectangle(Brushes::Blue, 1165.0, (float)10 + TaskHeight \* i, 90.0, TaskHeight); break;

case 5: gr->FillRectangle(Brushes::DarkBlue, 1165.0, (float)10 + TaskHeight \* i, 90.0, TaskHeight); break;

case 6: gr->FillRectangle(Brushes::Purple, 1165.0, (float)10 + TaskHeight \* i, 90.0, TaskHeight); break;

default: gr->FillRectangle(Brushes::Black, 1165.0, (float)10 + TaskHeight \* i, 90.0, TaskHeight); break;

}

gr->DrawRectangle(blackpen, 1165.0, (float)10 + TaskHeight \* i, 90.0, TaskHeight);

String^ drawString = System::Convert::ToString(CopyQ.pop().ID);

System::Drawing::Font^ drawFont = gcnew System::Drawing::Font("Arial", 16);

System::Drawing::SolidBrush^ drawBrush = gcnew System::Drawing::SolidBrush(System::Drawing::Color::Black);

gr->DrawString(drawString, drawFont, drawBrush, System::Drawing::RectangleF(1165.0, (float)10 + TaskHeight \* i, 90.0, TaskHeight));

}

}

if (!(qTask->isempty())) {

while (FreeProcCount(ProcArray, CountOfProc) >= (qTask->top().ProcCount)) {

Task CurrentTask = qTask->pop();

CountOfCurrentTasks++;

for (int j = 0; j < CurrentTask.ProcCount; j++) {

int CurrentID;

for (int k = 0; k < CountOfProc; k++) {

if (ProcArray[k].free) {

CurrentID = k;

break;

}

}

ProcArray[CurrentID].free = false;

ProcArray[CurrentID].ActiveTaskID = CurrentTask.ID;

ProcArray[CurrentID].CurrentStepCount = CurrentTask.StepCount;

ProcArray[CurrentID].MaxStepCount = CurrentTask.StepCount;

switch(CurrentTask.Color){

case 0: gr->FillRectangle(Brushes::Red, ProcArray[CurrentID].X, ProcArray[CurrentID].Y, 90, 65); break;

case 1: gr->FillRectangle(Brushes::Orange, ProcArray[CurrentID].X, ProcArray[CurrentID].Y, 90, 65); break;

case 2: gr->FillRectangle(Brushes::Yellow, ProcArray[CurrentID].X, ProcArray[CurrentID].Y, 90, 65); break;

case 3: gr->FillRectangle(Brushes::Green, ProcArray[CurrentID].X, ProcArray[CurrentID].Y, 90, 65); break;

case 4: gr->FillRectangle(Brushes::Blue, ProcArray[CurrentID].X, ProcArray[CurrentID].Y, 90, 65); break;

case 5: gr->FillRectangle(Brushes::DarkBlue, ProcArray[CurrentID].X, ProcArray[CurrentID].Y, 90, 65); break;

case 6: gr->FillRectangle(Brushes::Purple, ProcArray[CurrentID].X, ProcArray[CurrentID].Y, 90, 65); break;

default: break;

}

}

}

}

for (int i = 0; i < CountOfProc; i++) {

if (ProcArray[i].free) {

ProcArray[i].InactiveTactsCount++;

CountOfInactiveTacts++;

}

else {

ProcArray[i].CurrentStepCount--;

if (ProcArray[i].CurrentStepCount == 0) {

ProcArray[i].free = true;

ProcArray[i].ActiveTaskID = -1;

ProcArray[i].MaxStepCount = 0;

gr->FillRectangle(Brushes::Gray, ProcArray[i].X, ProcArray[i].Y, 90, 65);

}

}

}

TactsCount++;

label7->Text = System::Convert::ToString(TactsCount);

InactiveTactsPerProcessor = (double) CountOfInactiveTacts / CountOfProc;

}

catch (char \*str) {

System::String ^newStr = gcnew String(str);

label6->Text = newStr;

}

}

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

System::String ^str = "Всего тактов: " + System::Convert::ToString(TactsCount) + "\nТактов простоя (в среднем на процессор): " + System::Convert::ToString(InactiveTactsPerProcessor) + "\nОчередь заполнилась " + System::Convert::ToString(FullQTaskCount) + " раз(а)\nВыполнено задач: " + System::Convert::ToString(CountOfCurrentTasks);

System::Windows::Forms::DialogResult result;

result = MessageBox::Show(str,"Статистика");

}

};

}