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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кондратьев Я.С.

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)

[5. Эксперименты 8](#_Toc153312387)

[6. Заключение 9](#_Toc153312388)

[7. Литература 10](#_Toc153312389)

[8. Приложение 1 11](#_Toc153312390)

[9. Приложение 2 13](#_Toc153312391)

[10. Приложение 3 14](#_Toc153312392)

[11. Приложение 4 16](#_Toc153312393)

[12. Приложение 5 17](#_Toc153312394)

[13. Приложение 6 18](#_Toc153312395)

1. Введение

Цель данной лабораторной работы заключается в изучении структуры данных, называемой очередь, которая позволяет хранить элементы в порядке их поступления. В процессе работы мы рассмотрим различные способы представления и использования очередей и разработаем методы и программы для решения задач с их применением.

Очереди широко применяются в различных областях, таких как системы массового обслуживания и операционные системы. Они позволяют упорядочивать и обрабатывать элементы в соответствии с их поступлением.

Как пример использования очереди в данной лабораторной работе предлагается разработка имитационной системы однопроцессорной ЭВМ. Мы рассмотрим простую модель обслуживания задач, которая поможет нам ознакомиться с проблемами моделирования и анализа эффективности реальных вычислительных систем. Эта модель является лишь начальным этапом исследования, направленным на понимание основных принципов функционирования компьютерных 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.

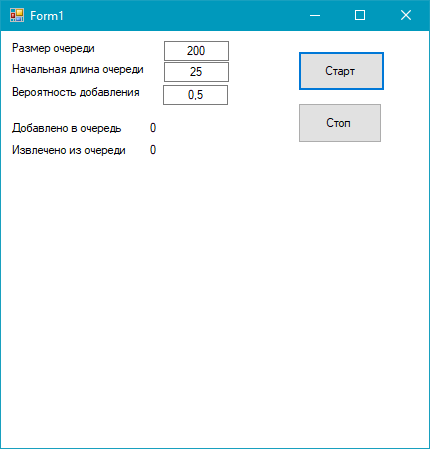
По результатам проводимых вычислительных экспериментов система имитации должна выводить информацию об условиях проведения эксперимента (интенсивность потока заданий, размер очереди заданий, производительность процессора, число тактов имитации) и полученные в результате имитации показатели функционирования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

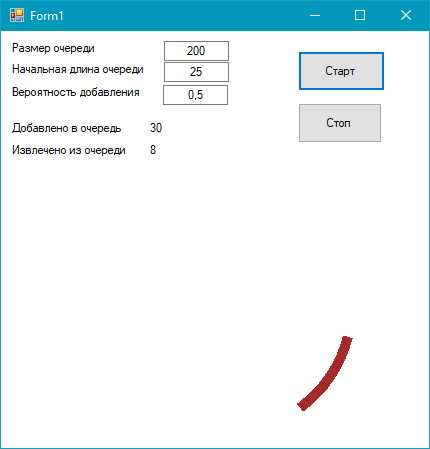
* количество поступивших в ВС заданий;
* количество тактов бездействия;
* среднее количество занятых процессоров;
* количество выполняемых задач;
* количество задач в очереди;
* количество выполненных задач.

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 выполнение программы до тех пор, пока не будет нажата кнопка «Старт».

Круговая диаграмма в правом нижнем углу отражает заполненность кольцевой очереди.

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:

int size – размер очереди

int start - индекс начала очереди

size\_t end – индекс конца очереди

size\_t count – количество элементов

T\* mas – массив размера size

public:

TQueue(int n = 0) - 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

TQueue<T>& operator=(const TQueue<T>& q) 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

1. Эксперименты

Эксперименты проводились на ПК с следующими параметрами:

1. Операционная система: Windows 10
2. Процессор: Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz 3.60 GHz
3. Версия Visual Studio: 2022

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

1. Заключение

В ходе работы был создан экземпляр класса TQueue, представляющий кольцевую очередь с эффективными алгоритмами изменения. Кроме того, было разработано приложение на платформе WinForms с использованием языка программирования C++, которое эмулирует систему обслуживания потока задач на компьютере.

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  
#pragma once

#include <iostream>

template <class T>

class TQueue {

protected:

int size;

int start; // индекс начала очереди

int end;

int count; // количество элементов в очереди

T\* mas;

public:

TQueue(int n = 0) {

if (n <= 0)

throw - 1;

size = n;

start = 0;

end = size;

count = 0;

mas = new T[size];

}

TQueue(TQueue <T>& q) {

start = q.start;

end = q.end;

count = q.count;

size = q.size;

mas = new T[size];

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

mas[i] = q.mas[i];

}

~TQueue() {

if (mas == nullptr)

delete[] mas;

mas = nullptr;

}

void Push(T a) {

if (IsFull())

throw - 1;

int index = (start + count++) % size;

mas[index] = a;

end = (end + 1) % size;

}

T Get() {

if (IsEmpty())

throw - 1;

T tmp = mas[start];// элемент помещается в tmp

mas[start] = T();

start = (start + 1) % size; // изменение значения полей

count--;

end = (end + 1) % size;

return tmp;

}

bool IsFull() {

return count == size;

}

bool IsEmpty() {

return count == 0;

}

int GetStart() { return start; }

int GetSize() { return size; }

int GetCount() { return count; }

int GetEnd() { return end; }

TQueue& operator=(const TQueue<T>& q) { // присваивает одну очередь другой

if (this == &q)

return \*this;

if (mas != nullptr)

delete[] mas;

size = q.size;

mas = new T[size];

for (int i = 0; i < size; i++) // происходит копирование обратно

mas[i] = q.mas[i];

count = q.count;

start = q.start;

end = q.end;

return \*this;

}

//операторы вводы и выводы

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

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

ostr << q.mas[i] << std::endl;

}

return ostr;

}

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

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

istr >> q.mas[i];

}

return istr;

}

};

1. Приложение 2

test\_main.cpp  
#include <gtest.h>

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

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

return RUN\_ALL\_TESTS();

}

1. Приложение 3

test\_queue.cpp  
#include "MyQueue.h"

#include <gtest.h>

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\_start)

{

TQueue<int> s1(2);

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

}

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);

}

1. Приложение 4

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:\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.GetStart() << endl;

break;

case 6:

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

break;

default:

break;

}

cout << q << endl;

}

}

catch (exception e) {

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

}

}

1. Приложение 5

MyForm.cpp  
#include "MyForm.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThread]

void Main()

{

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

CppWinForm1::MyForm form;

Application::Run(%form);

}

1. Приложение 6

Queue\_experiments.cpp  
#include <iostream>

#include <chrono>

#include <random>

#include "../../queue/include/MyQueue.h"

auto average\_test(size\_t size, size\_t iterations = 10) {

int max\_random = 10000;

int min\_random = -10000;

long long average\_time\_add = 0;

long long average\_time\_del = 0;

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

TQueue<int> queue(size);

std::cout << "#" << i << " Adding" << std::endl;

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

int c1 = min\_random + std::rand() % static\_cast<int>(max\_random - min\_random + 1);

std::chrono::steady\_clock::time\_point begin, end;

begin = std::chrono::steady\_clock::now();

queue.Push(c1);

end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

average\_time\_add += elapsed\_ms.count();

std::cout << i + 1 << '\t' << elapsed\_ms.count() << std::endl;

}

std::cout << std::endl << "#" << i << " Delliting" << std::endl;

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

std::chrono::steady\_clock::time\_point begin, end;

begin = std::chrono::steady\_clock::now();

queue.Get();

end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

average\_time\_add += elapsed\_ms.count();

std::cout << i + 1 << '\t' << elapsed\_ms.count() << std::endl;

}

}

std::cout << "Average time add: " << average\_time\_add / static\_cast<double>(iterations \* size) << std::endl;

std::cout << "Average time del: " << average\_time\_add / static\_cast<double>(iterations \* size) << std::endl;

return;

}

int main(int argc, char\*\* arhv)

{

auto begin = std::chrono::steady\_clock::now();

average\_test(150);

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

std::cout << "Execution time: " << elapsed\_ms.count();

}