Министерство науки и высшего образования РФ

Федеральное государственное автономное образовательное   
учреждение высшего профессионального образования

«Нижегородский государственный университет им. Н.И. Лобачевского»

Институт информационных технологий математики и механики

**Отчет по учебной практике**

**Лабораторная работа № 4**

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

**Выполнил**:студент группы 381703-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Полегайко В.А.

Подпись

**Научный руководитель**:

доцент каф. МОСТ, к.т.н.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сысоев А.В.

Подпись

**Содержание**

[Введение 3](#__RefHeading__3039_1807997502)

[Постановка задачи 4](#__RefHeading__3041_1807997502)

[Руководство пользователя 5](#__RefHeading__3043_1807997502)

[Руководство программиста 6](#__RefHeading__3045_1807997502)

[Описание структуры программы 7](#__RefHeading__3047_1807997502)

[Описание алгоритмов 8](#__RefHeading__3049_1807997502)

[Заключение 9](#__RefHeading__3051_1807997502)

[Литература 10](#__RefHeading__3053_1807997502)

[Приложение 11](#__RefHeading__3055_1807997502)

# **Введение**

Динамические структуры занимают важное место в современной теории программирования. Динамическая структура есть математическая структура, которой соответствует частично-упорядоченное (по включению) базовое множество ***М,*** элементы которого являются структурами данных. При этом отношения включения формируются операциями преобразования структуры данных.

Очередь — динамическая структура

S=(Mi, P1, P2)

в которой:

* каждый элемент — структура,
* в любой момент существует только один конкретный элемент из  ***М***,
* элементы частично упорядочены по включению.

P1 – отношение следования, порождаемое операцией вставки,

P2 – отношение следования, порождаемое операцией исключение.

P1, P2 организованы по правилу FIFO или «первый вошел — первый вышел».

Лабораторная работа направлена на практическое освоение динамической структуры данных Очередь. С этой целью в лабораторной работе изучаются различные варианты структуры хранения очереди и разрабатываются методы и программы решения задач с использованием очередей.

В качестве области приложений выбрана тема эффективной организации выполнения потока заданий на вычислительных системах. Очередь характеризуется таким порядком обработки значений, при котором вставка новых элементов производится в конец очереди, а извлечение – из начала. Подобная организация данных широко встречается в различных приложениях. В качестве примера использования очереди предлагается задача разработки системы имитации однопроцессорной ЭВМ. Рассматриваемая в рамках лабораторной работы схема имитации является одной из наиболее простых моделей обслуживания заданий в вычислительной системе и обеспечивает тем самым лишь начальное ознакомление с проблемами моделирования и анализа эффективности функционирования реальных вычислительных систем.

# **Постановка задачи**

Для вычислительной системы (ВС) с одним процессором и однопрограммным последовательным режимом выполнения поступающих заданий требуется разработать программную систему для имитации процесса обслуживания заданий в ВС. 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 отказов в обслуживании заданий из-за переполнения очереди;
* среднее количество тактов выполнения заданий;
* количество тактов простоя процессора из-за отсутствия в очереди заданий дляобслуживания.

# **Руководство пользователя**

Данная программа написана в среде разработки Microsoft Visual Studio 2017 на языке С++ и является консольным приложением. При запуске приложения, будет предложено инициализировать размер очереди, число тактов имитации, интенсивность потока задач и производительность процессора, см. рис.1.



Рис. 1. Инициализация данных

После инициализации, произойдет вывод результатов работы процессора, см. рис.2.



Рис.2. Вывод результатов работы.

# **Руководство программиста**

Программа написана при помощи трех классов Queue, TProc, TJobStream.

Класс Queue содержит следующие приватные поля:

1. int \*pMem – Указатель, отвечающий за хранение элементов в очереди.
2. int MemSize – размер очереди.
3. int Hi – Указатель на последний элемент очереди
4. int Lo – Указатель на первый элемент очереди
5. int DataCount - Кол-во элементов находящихся в очереди

Класс Queue содержит следующие методы:

1. Queue(int Size = 10) – Конструктор класса.
2. ~Queue(void) – Деструктор.
3. Queue (const Queue &tmp) – Конструктор копирования.
4. Queue &operator=(const TQueue &tmp) – Оператор присваивания.
5. int IsEmpty() – Метод проверки пустоты очереди.
6. int IsFull() – Метод проверки полноты очереди.
7. void Put(int tmp) – Функция кладет в конец очереди элемент.
8. int Get() – Функция выдачи первого элемента из очереди.

Класс TProc содержит следующие приватные поля:

1. double ProcRate - производительность процессора.
2. int JobId – ID выполняемого задания

Класс TProc содержит следующие методы:

1. TProc(double Rate) – Конструктор класса.
2. int IsProcBusy(void) – Метод возвращает значение 0, если процессор свободен, и 1, если процессор занят.
3. int RunNewJob() – Метод запуска работы процессора.
4. ~TProc(){} - Деструктор класса.

Класс TJobStream содержит следующие приватные поля:

double JobIntens – Интенсивность потока задач.

Класс TJobStream содержит следующие методы:

TJobStream(double JobIntens) — Конструктор класса.

int GetNewJob(void) - Метод возвращает значение 0, если поступило новое задание, и 1, если нет.

## **Описание структуры программы**

Программа содержит один основной проект - QueueWithTests3.

Проект QueueWithTests3 содержит в себе модули Queue.h, TProc.h, TJobStream.h. Модули содержат в себе объявление классов Queue, TProc и TJobStream соответственно. Реализация методов классов содержатся в отдельном файле source QueueWithTests3.cpp.

## **Описание алгоритмов**

**Алгоритмы класса Queue**

1. Queue(int Size = 10) – Конструктор-инициализатор, с начальным значением размерности 10. Выделяем память для хранения элементов очереди, так же присваивает некоторые значения по умолчанию.
2. int IsEmpty() – Метод проверки пустоты очереди. Если значение переменной DataCount будет равно нулю, значит очередь пуста, в этом случае метод вернет 0, в ином случае 1.
3. int IsFull() – Метод проверки полноты очереди. Если значение переменной DataCount будет равно переменной MemSize, значит очередь полная, вернет в таком случае 0, иначе вернет 1.
4. void Put(int tmp) – Функция кладет в конец очереди элемент и увеличивает DataCount.
5. int Get() – Метод возвращает первый элемент очереди, уменьшает DataCount.

**Алгоритмы класса TProc**

1. TProc(double Rate) - Конструктор-инициализатор. Присваивает ProcRate значение Rate\*RAND\_MAX, JobId присваивается 0.
2. int IsProcBusy(void) – присваивать переменной a случайное число и сравнивает его с ProcRate. Возвращает 0, если a < ProcRate, тогда процессор свободен, иначе возвращает 1, процессор занят.
3. int RunNewJob(int \_JobId) – если процессор свободен, присваивает JobId значение \_JobId.

**Алгоритмы класса TJobStream**

1. TJobStream(double Intens) – Конструктор-инициализатор. Присваивает JobIntens случайное значение.
2. int GetNewJob(void) – присваивает a случайное число, если оно меньше JobIntens, то возвращает 0, поступило новое задание, иначе возвращает 1, нового задания нет.

# **Заключение**

Данная лабораторная работа познакомила нас с методом хранения информации «очередь», алгоритмами создания очереди, помещение и извлечение элементов на примере имитации работы процессора.

# **Литература**

1. Столлингс, В. Структурная организация и архитектура компьютерных систем, 5-е изд.: Пер. с англ. — М.: Издательский дом «Вильямс», 2002. — 896 с.: ил. — Парал. тит. англ.
2. Брайан Керниган, Деннис Ритчи «Язык программирования Си».
3. Герберт Шилдт - Полный справочник по C
4. Вирт Н. «Алгоритмы и структуры данных».
5. Кормен Т., Лейзерсон Ч., Ривест Р. «Алгоритмы: построение и анализ».

# **Приложение**

**Модуль Queue.h**

#pragma once

#define MaxMemSize 10

class Queue

{

int \*pMem;

int MemSize;

int Hi;

int Li;

int DataCount;

public:

Queue(int Size = MaxMemSize)

{

MemSize = Size;

pMem = new int[MemSize];

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

{

pMem[i] = 0;

}

Hi = 0;

Li = 0;

DataCount = 0;

};

~Queue(void)

{

delete[] pMem;

};

Queue(const Queue &tmp)

{

MemSize = tmp.MemSize;

Hi = tmp.Hi;

Li = tmp.Li;

DataCount = tmp.DataCount;

pMem = new int[MemSize];

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

{

pMem[i] = tmp.pMem[i];

}

};

Queue &operator= (Queue tmp)

{

if (MemSize != tmp.MemSize)

{

if (MemSize != 0)

{

delete[] pMem;

}

MemSize = tmp.MemSize;

pMem = new int[MemSize];

}

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

{

pMem[i] = tmp.pMem[i];

}

Hi = tmp.Hi;

Li = tmp.Li;

DataCount = tmp.DataCount;

};

int IsEmpty()

{

if (DataCount == 0)

return 0; //Очередь пуста

else return 1; // Очередь не пуста

}

int IsFull()

{

if (DataCount == MemSize)

return 0; //Очередь полна

else return 1; // Очередь не полна

}

void Put(int tmp)

{

pMem[Hi++] = tmp;

if (Hi == MemSize)

Hi = 0;

DataCount++;

}

int Get()

{

int a = pMem[Li];

DataCount--;

if (DataCount == 0)

{

Hi = 0;

Li = 0;

}

else

{

Li++;

if (Li == MemSize)

Li = 0;

}

return a;

}

};

**Модуль TProc.h**

#pragma once

#include <cstdlib>

class TProc

{

private:

double ProcRate;

int JobId;

public:

TProc(double Rate)

{

ProcRate = Rate \* RAND\_MAX;

JobId = 0;

};

int IsProcBusy(void)

{

double a = std::rand() % RAND\_MAX;

if (a < ProcRate)

{

return 0;//процессор свободен

}

else

{

return 1; //процессор занят

}

};

int RunNewJob(int \_JobId)

{

if (IsProcBusy() == 0)

{

JobId = \_JobId;

};

return JobId;

};

~TProc() {};

};

**Модуль TJobStream.h**

#pragma once

#include <cstdlib>

#include <ctime>

class TJobStream

{

private:

double JobIntens;

public:

TJobStream(double Intens)

{

std::srand(unsigned(std::time(0)));

JobIntens = Intens \* RAND\_MAX;

};

int GetNewJob(void)

{

double a = std::rand() % RAND\_MAX;

if (a < JobIntens)

{

return 0;//поступило новое задание

}

else

{

return 1; //нового задания нет

}

};

~TJobStream() {};

};

**Модуль QueueWithTests3.cpp**

#include "pch.h"

#include <stdio.h>

#include "Queue.h"

#include "iostream"

#include "stdlib.h"

#include "stdio.h"

#include "TJobStream.h"

#include "TProc.h"

using namespace std;

int main()

{

int n;

cout << "Enter maximal queue size" << endl;

cin >> n;

Queue Q(n);

/\*cout << "Control pustoty " << Q.IsEmpty() << endl;

cout << "Control polnoty " << Q.IsFull() << endl;

for (int i=0; i<5; i++)

{

Q.Put(i+2\*i);

}

cout << "Control pustoty " << Q.IsEmpty() << endl;

cout << "Control polnoty " << Q.IsFull() << endl;

int i;

while (Q.IsFull()!=0)

{

Q.Put(4);

}

cout << "Control pustoty " << Q.IsEmpty() << endl;

cout << "Control polnoty " << Q.IsFull() << endl;

while (Q.IsEmpty() == 1)

{

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

}

cout << "Control pustoty " << Q.IsEmpty() << endl;

cout << "Control polnoty " << Q.IsFull() << endl;\*/

int count;

cout << "Enter count of tact" << endl;

cin >> count;

int number = 0;

int counttask = 0;

int countreject = 0;

int countempty = 0;

int oldnum = 0;

int \*listnum;

int \*listtact;

listnum = new int[count];

listtact = new int[count];

cout << "Enter q1 from 0 to 1" << endl;

double q1;

cin >> q1;

TJobStream TJ(q1);

/\*cout << RAND\_MAX << endl;

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

{

cout << "Is job here?" << TJ.GetNewJob() << endl;

}\*/

cout << "Enter q2 from 0 to 1" << endl;

double q2;

cin >> q2;

TProc TP(q2);

/\*for (int i = 0; i < 100; i++)

{

cout << "Is job here?" << TP.RunNewJob() << endl;

}\*/

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

{

if (TJ.GetNewJob() == 0)

{

if (Q.IsFull() != 0)

{

Q.Put(++number);

}

else

{

cout << "Queue is full" << endl;

countreject++;

};

};

if (TP.IsProcBusy() == 0)

{

if (Q.IsEmpty() == 1)

{

int job = TP.RunNewJob(Q.Get());

}

else

{

cout << "Queue is empty" << endl;

countempty++;

}

}

if (oldnum == number)

{

listtact[counttask - 1]++;

}

else

{

oldnum = number;

listnum[counttask] = number;

listtact[counttask++] = 1;

};

cout << "Number " << number << endl;

};

cout << "Count of full queue " << countreject \* 100 / count << "%" << endl;

cout << "Count of empty queue " << countempty \* 100 / count << "%" << endl;

cout << "Count of task " << counttask << endl;

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

{

cout << "Number of task " << listnum[i] << " Count tact " << listtact[i] << endl;

}

system("pause");

return 0;

}