Министерство Образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Нижегородский Государственный Университет им. Н.И. Лобачевского» «Институт Информационных Технологий Математики и Механики»

**Отчет**

**По лабораторной работе №4**

**«Система имитации потока задач»**

Выполнил:

Студент группы 0826-1

Смертин Д. С.

Проверил:

Доцент кафедры МОСТ

Сысоев А. В.

г. Нижний Новгород

2016 г.

Оглавление

[Введение 3](#_Toc470362202)

[Постановка задачи 4](#_Toc470362203)

[Руководство пользователя 5](#_Toc470362204)

[Руководство программиста 7](#_Toc470362205)

[Цикл работы 10](#_Toc470362206)

[Заключение 11](#_Toc470362207)

[Литература 12](#_Toc470362208)

[Приложение 13](#_Toc470362209)

# Введение

В лабораторной работе были реализованы следующие классы: список, очередь и класс системы имитации потока задач (далее СИПЗ).

Список (англ. list) -- это абстрактный тип данных, представляющий собой упорядоченный набор значений, в котором некоторое значение может встречаться более одного раза. Экземпляр списка является компьютерной реализацией математического понятия конечной последовательности.

Очередь -- абстрактный тип данных с дисциплиной доступа к элементам «первый пришёл -- первый вышел» (FIFO, First In -- First Out). Добавление элемента возможно лишь в конец очереди, выборка -- только из начала очереди, при этом выбранный элемент из очереди удаляется. Очередь была реализована с помощью списка.

СИПЗ состоит из двух основных объектов: процессора и задач. Процессор определяют количество тактов и очередь определенной длинны. Задачу определяют пороги создания и завершения задачи.

# Постановка задачи

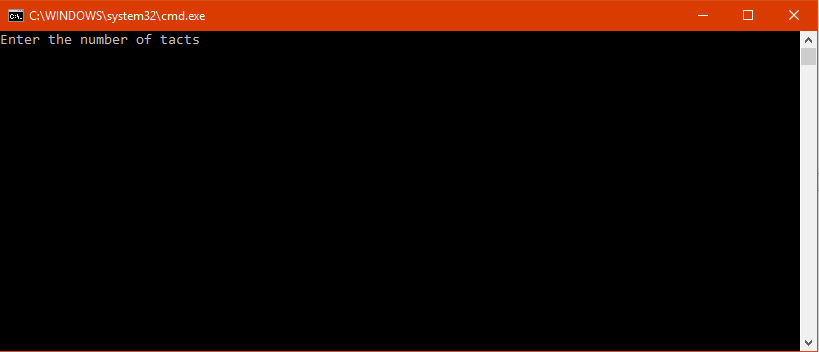
Цель данной лабораторной работы – разработать программу, в которой будет реализована СИПЗ. Для этого необходимо реализовать класс очередь и класс СИПЗ. Результатом выполнения программы должна являться статистика, включающая в себя следующие данные:

1. Количество задач, которые были завершены;
2. Количество задач, которые остались в очереди задач;
3. Количество задач, которые не попали в очередь, из-за ее переполнения;
4. Количество тактов простоя процессора.

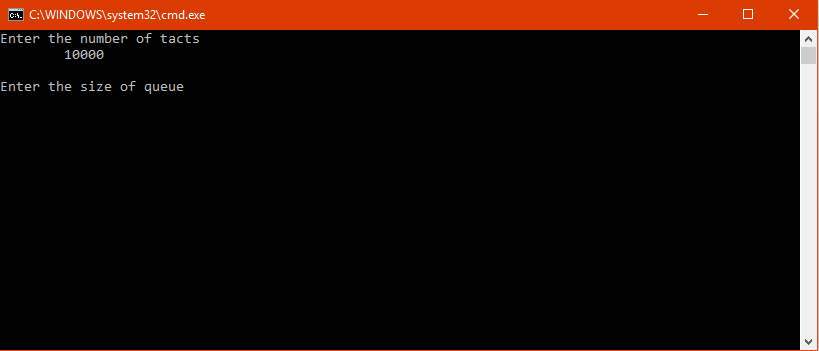
К программе (к каждому реализованному классу) нужно написать тесты, которые будут подтверждать корректность ее работы.

# Руководство пользователя

Для запуска программы перейдите в папку «LabWork4», далее в папку «Debug». В данной папке найдите и запустите «LabWork4.exe». На экране появится консоль:

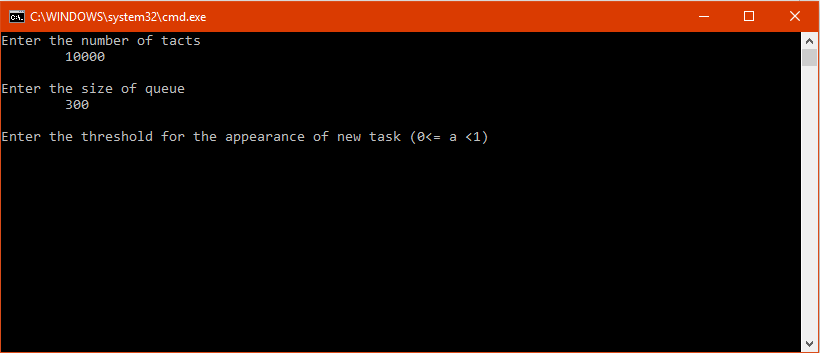


Нужно ввести количество тактов процессора.

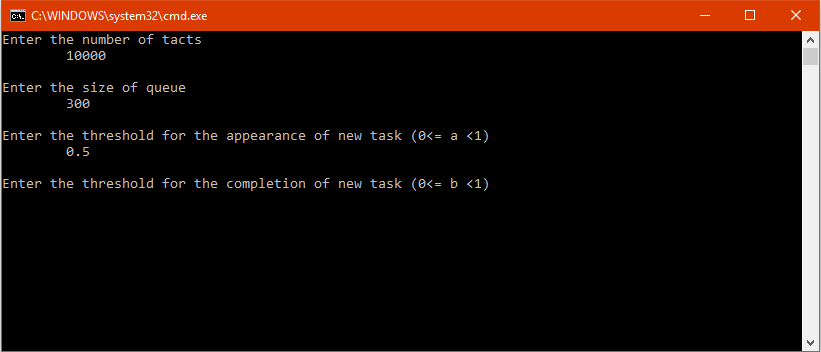


Нужно ввести размер очереди задач.

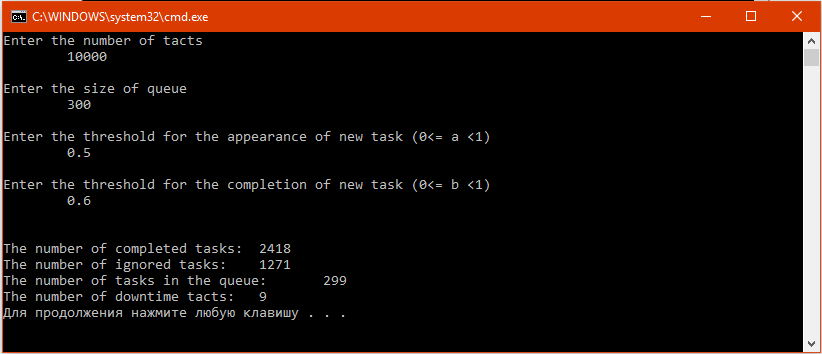
Максимальное количество тактов и максимальная длина очереди равны 10000 ед.



Нужно ввести порог появления новой задачи



Нужно ввести порог завершения текущей задачи



После этого программа выведет статистику работы имитированного процессора.

# Руководство программиста

Структура программы

Лабораторная работа состоит из 3 проектов: «LabWork4», «Tests» и «gtest».

«LabWork4»

Проект «LabWork4» состоит из пяти заголовочных файлов: «CImCPU.h», «CList.h», «CQueue.h» и «SNode.h» --, в которых реализованы соответственно классы СИПЗ, список (шаблонный), очередь (шаблонный) и структура звена списка. Также в этом проекте есть файлы исходного кода: «CImCPU.cpp», в котором реализованы методы класса СИПЗ и «Main.cpp» который содержит функцию «int main()».

«CList.h»

В шаблонном классе список объявлены поля:

1. SNode<T>\* pFirst;
2. size\_t count;

Методы, объявленные (и реализованные) в списке:

1. CList ()
2. ~CList ()
3. bool IsEmpty ()
4. bool IsFull ()
5. void InsFirst (T val)
6. void InsLast (T val)
7. void Insert (T val, size\_t pos)
8. void DelFirst ()
9. void DelLast ()
10. void Delete (size\_t pos)
11. size\_t GetCount()
12. T GetVal(size\_t pos)
13. void ReVal(T val, size\_t pos)

«CQueue.h»

«CQueue.h» содержит класс очередь.

В классе объявлены поля:

1. CList<T> mem;
2. size\_t top;
3. size\_t size;

Методы:

1. CQueue()
2. CQueue(size\_t s)
3. bool IsEmpty()
4. bool IsFull()
5. void Put(T val)
6. T Get()
7. T GetVal()

«CQueue.h»

«CQueue.h» содержит класс СИПЗ.

Поля класса:

1. unsigned int numTacts;
2. unsigned int sizeQueue;
3. unsigned int tact;
4. bool work;
5. double markNewTask;
6. double markCompletionTask;
7. unsigned int numCopmletedTask;
8. unsigned int numTaskQueue;
9. unsigned int numIgnoredTasks;
10. unsigned int numDowntimeTacts;

Методы:

1. CImCPU(unsigned int nTacts, unsigned int sQueue, double mNewTask, double mCompletionTask);
2. CImCPU();
3. void Process();
4. void Statistics();

«Tests»

«Tests» содержит следующие файлы: «CImCPU.h», «CImCPU.cpp», «CImCPU\_test.cpp», «CList\_test.cpp», «CQueue\_test.cpp», «test\_main.cpp». Файлы содержат тесты, проверяющие классы на корректную работу. К ним подключена библиотека <gtest.h>.

# Цикл работы

Процессор имеет t тактов. Каждый такт:

1. Может появиться новая задача:
   1. 0<а<1 – порог появления новой задачи;
   2. Генерируется случайное 0<α<1;
   3. Если α>а, то появляется новая задача.
2. Если появилась новая задача, то она помещается в очередь;
3. Если очередь заполнена, то задача игнорируется;
4. Если процессор свободен, то он берет задачу из очереди;
5. Если задача поставлена до текущего такта, то проверяется ее завершение:
   1. 0<б<1 – порог завершения задачи;
   2. Генерируется случайное 0<β<1;
   3. Если β >б, то задача завершается.
6. Если номер такта равен t, то процессор завершает работу.

По окончанию работы процессора выводится статистика:

1. Количество задач, которые были завершены (в том числе текущая);
2. Количество задач, которые остались в очереди задач;
3. Количество задач, которые не попали в очередь, из-за ее переполнения;
4. Количество тактов простоя процессора.

# Заключение

В процессе выполнения лабораторной работы изучены такие структуры данных, как список и очередь, выполнена их реализация в виде класса.

Также изучены алгоритм работы имитированного процессора.

Была реализована программа, имитирующая систему потока задач.

К программе были написаны тесты, подтверждающие корректность её работы.

# Литература

1. Герберт Шилдт. Полный справочник по C++ = C++: The Complete Reference. — 4-е изд. — М.: Вильямс, 2011. — С. 800. — ISBN 978-5-8459-0489-8.
2. Бьёрн Страуструп. Программирование: принципы и практика использования C++, исправленное издание = Programming: Principles and Practice Using C++. — М.: Вильямс, 2011. — С. 1248. — ISBN 978-5-8459-1705-8.
3. Майкл Мейн, Уолтер Савитч. Структуры данных и другие объекты в C++ = Data Structures and Other Objects Using C++. — 2-е изд. — М.: Вильямс, 2002. — 832 с. — ISBN 0-201-70297-5.

# Приложение

«SNode.h»

template< class T >

struct SNode

{

T value;

SNode\* pNext;

};

«CList.h»

#pragma once

#include "SNode.h"

#define TNode SNode

#define TList Clist

#define MaxL 10000

template <class T>

class CList

{

SNode<T>\* pFirst;

size\_t count;

public:

CList() : pFirst(nullptr), count(0) {}

~CList()

{

while (IsEmpty() != true)

DelLast();

}

bool IsFull()

{

return count == MaxL;

}

bool IsEmpty()

{

return count == 0;

}

void InsFirst(T val)

{

if (IsFull())

throw val;

else

{

SNode<T> \*p = new SNode<T>;

p->value = val;

p->pNext = pFirst;

pFirst = p;

count++;

}

}

void InsLast(T val)

{

if (IsFull())

throw val;

if (IsEmpty())

InsFirst(val);

else

{

SNode<T> \*p = pFirst;

SNode<T> \*temp = new SNode<T>;

for (size\_t i = 0; i < count - 1; i++)

p = p->pNext;

temp->value = val;

temp->pNext = nullptr;

if (p != nullptr)

p->pNext = temp;

else

pFirst = temp;

count++;

}

}

void Insert(T val, size\_t pos)

{

if (IsFull())

throw val;

if (pos > count)

throw pos;

if (pos == 0)

InsFirst(val);

else

if (pos == count)

InsLast(val);

else

{

SNode<T> \*temp = new SNode<T>;

temp->value = val;

SNode<T> \*p = pFirst;

for (size\_t i = 0; i < pos - 1; i++)

p = p->pNext;

temp->pNext = p->pNext;

p->pNext = temp;

count++;

}

}

void DelFirst()

{

if (IsEmpty())

throw 0;

SNode<T> \*p = pFirst;

pFirst = p->pNext;

delete p;

count--;

}

void DelLast()

{

if (IsEmpty())

throw 0;

SNode<T> \*p = pFirst;

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

p = p->pNext;

delete p;

count--;

}

void Delete(size\_t pos)

{

if (pos > count)

throw pos;

if (IsEmpty())

throw 0;

else

if (pos == 0)

DelFirst();

else

if (pos == count -1)

DelLast();

else

{

SNode<T> \*p = pFirst;

SNode<T> \*past = pFirst;

for (size\_t i = 0; i < pos-1; i++)

past = past->pNext;

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

p = p->pNext;

SNode<T> \*temp = p->pNext;

past->pNext = temp;

delete p;

count--;

}

}

size\_t GetCount()

{

return count;

}

T GetVal(size\_t pos)

{

if (pos > count)

throw pos;

SNode<T> \*p = pFirst;

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

p = p->pNext;

T res = p->value;

return res;

}

void ReVal(T val, size\_t pos)

{

/\*Delete(pos);

Insert(val, pos);\*/

if (pos > count)

throw pos;

SNode<T>\* p = pFirst;

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

p = p->pNext;

p->value = val;

}

};

«CQueue.h»

#pragma once

#include "SNode.h"

#include "CList.h"

#define MaxQ 10000

template <class T>

class CQueue

{

CList<T> mem;

size\_t top;

size\_t size;

public:

CQueue() : top(-1), size(MaxQ) {}

CQueue(size\_t s) : top(-1)

{

if (s > MaxQ || s < 1)

throw s;

size = s;

}

bool IsEmpty()

{

return top == (-1);

}

bool IsFull()

{

return top == (size - 1);

}

void Put(T val)

{

if (IsFull())

throw val;

mem.InsFirst(val);

top++;

}

T Get()

{

if (IsEmpty())

throw 0;

T val;

val = mem.GetVal(mem.GetCount()-1);

mem.DelLast();

top--;

return val;

}

T GetVal()

{

if (IsEmpty())

throw 0;

T val;

val = mem.GetVal(mem.GetCount() - 1);

return val;

}

};

«CImCPU.h»

#pragma once

#include "CQueue.h"

#define t 10000

class CImCPU

{

unsigned int numTacts;

unsigned int sizeQueue;

unsigned int tact;

bool work;

double markNewTask;

double markCompletionTask;

unsigned int numCopmletedTask;

unsigned int numTaskQueue;

unsigned int numIgnoredTasks;

unsigned int numDowntimeTacts;

public:

CImCPU(unsigned int nTacts, unsigned int sQueue, double mNewTask, double mCompletionTask);

CImCPU();

void Process();

void Statistics();

};

«CImCPU.cpp»

#include "CImCPU.h"

#include <ctime>

#include <iostream>

using namespace std;

CImCPU::CImCPU() : tact(0), numCopmletedTask(0), numTaskQueue(0), numIgnoredTasks(0), numDowntimeTacts(0), work(false)

{

numTacts = t;

sizeQueue = MaxQ;

markNewTask = 0.5;

markCompletionTask = 0.5;

}

CImCPU::CImCPU(unsigned int nTacts, unsigned int sQueue, double mNewTask, double mCompletionTask) : tact(0), numCopmletedTask(0), numTaskQueue(0), numIgnoredTasks(0), numDowntimeTacts(0), work(false)

{

if (nTacts > t || sQueue > MaxQ || mCompletionTask >= 1 || mNewTask >= 1 || nTacts < 0 || sQueue < 0 || mCompletionTask < 0 || mNewTask < 0)

throw "Incorrect values";

numTacts = nTacts;

sizeQueue = sQueue;

markNewTask = mNewTask;

markCompletionTask = mCompletionTask;

}

void CImCPU::Process()

{

CQueue<unsigned int> queueTasks(sizeQueue);

srand(time(0));

for (tact; tact < numTacts; tact++)

{

if (work == true)

if ((1.0\*(rand() % 10)) / 10 > markCompletionTask)

{

work = false;

numCopmletedTask++;

}

if ((1.0\*(rand() % 10)) / 10 > markNewTask)

if (queueTasks.IsEmpty())

if (work == false)

{

work = true;

continue;

}

else

{

queueTasks.Put(tact);

continue;

}

else

if (queueTasks.IsFull())

{

numIgnoredTasks++;

continue;

}

else

{

queueTasks.Put(tact);

continue;

}

else

if (work == false)

if (queueTasks.IsEmpty())

{

numDowntimeTacts++;

continue;

}

else

{

queueTasks.Get();

work = true;

continue;

}

}

if (work==true)

numCopmletedTask++;

while (!queueTasks.IsEmpty())

{

queueTasks.Get();

numTaskQueue++;

}

}

void CImCPU::Statistics()

{

cout << "The number of completed tasks:\t" << numCopmletedTask << "\n";

cout << "The number of ignored tasks:\t" << numIgnoredTasks << "\n";

cout << "The number of tasks in the queue:\t" << numTaskQueue << "\n";

cout << "The number of downtime tacts:\t" << numDowntimeTacts << "\n";

}

«Main.cpp»

#include <iostream>

#include "CImCPU.h"

using namespace std;

void main()

{

unsigned int nTacts;

unsigned int sQueue;

double mNewTask;

double mCompletionTask;

setlocale(LC\_ALL, "Russian");

cout << "Enter the number of tacts\n\t";

cin >> nTacts;

cout << "\n";

cout << "Enter the size of queue\n\t";

cin >> sQueue;

cout << "\n";

cout << "Enter the threshold for the appearance of new task (0<= a <1)\n\t";

cin >> mNewTask;

cout << "\n";

cout << "Enter the threshold for the completion of new task (0<= b <1)\n\t";

cin >> mCompletionTask;

cout << "\n\n";

CImCPU ImCPU(nTacts, sQueue, mNewTask, mCompletionTask);

//CImCPU ImCPU;

setlocale(LC\_ALL, "Russian");

ImCPU.Process();

ImCPU.Statistics();

}