|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

«Моделирование работы информационного центра»

Студент ИУ7-68Б(В) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Н. Бахолдин

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель темы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И.В. Рудаков

(Подпись, дата) (И.О.Фамилия)

*2020 г*

Необходимо создать концептуальную модель в терминах СМО, определить эндогенные и экзогенные переменные и уравнения следующей системы :

В информационный центр приходят клиенты через интервал времени 10 +- 2 минуты. Если все три имеющихся оператора заняты, клиенту отказывают в обслуживании. Операторы имеют разную производительность и могут обеспечивать обслуживание среднего запроса пользователя за 20 +- 5; 40 +- 10; 40 +- 20. Клиенты стремятся занять свободного оператора с максимальной производительностью. Полученные запросы сдаются в накопитель. Откуда выбираются на обработку. На первый компьютер запросы от 1 и 2-ого операторов, на второй – запросы от 3-его. Время обработки запросов первым и 2-м компьютером равны соответственно 15 и 30 мин. Промоделировать процесс обработки 300 запросов. За единицу системного времени выбрать 0,01 минуты.

Переменные и уравнения имитационной модели.

Эндогенные переменные: время обработки задания i-ым оператором, время решения этого задания j-ым компьютером.

Экзогенные переменные: число обслуженных клиентов и число клиентов получивших отказ.



Декомпозируем задачу и построим модель СМО данной системы. Обработка заявок в информационном центре проводится в 3 этапа:

1. Принятие заявок одним из операторов.
2. Обработка заявок операторами и передача в накопитель.
3. Обработка заявок которые выбирают компьютеры из накопителя.

На первом этапе необходимо смоделировать процесс прихода заявок в интервале 10 +- 2 минуты и принятии или не принятии заявки оператором, то есть необходимо построить многоканальную СМО с отказами. В качестве потока заявок рассматриваются пришедшие клиенты, а в качестве каналов обслуживания - операторы. По условию задачи заявки приходят в интервале 10 +- 2, для моделирования подойдет равномерный закон распределения на интервале от 8 до 12 .

На втором этапе операторы получают заявки клиентов в порядке наибольшей производительности, что означает : очередную заявку получит тот оператор у которого производительность выше, чем у других( по смыслу это значит, что заявку получит тот оператор который на текущий момент( момент прихода заявки от клиента) обработал больше всех заявок по сравнению с другими операторами ). По условию задачи время обработки каждой полученной заявки у трех операторов разная и составляет: 20 +- 5; 40 +- 10; 40 +- 20, для моделирования подойдет равномерный закон распределения на интервалах : для первого оператора от 15 до 25, для второго от 30 до 50 и для третьего от 20 до 60. После окончания обработки заявки оператор помещает их в накопитель по следующему правилу: обработанные заявки от первого и второго оператора помещаются в накопитель первого компьютера, третьего оператора в накопитель второго компьютера.

На третьем этапе заявки обрабатывают компьютеры с фиксированным временем обработки одной заявки для первого компьютера время обработки фиксировано и составляет 15 минут, а для второго 30. В том случае если в ходе моделирования время моделирования СМО не задается, то третий этап сводится к следующему : общее количество заявок обработанных компьютерами вычисляется как количество заявок находящиеся в накопителе. В программной реализации данной модели СМО есть опция задавать время моделирования СМО или нет. В случае выбора опции моделирования СМО со временем, третий этап сводится к вычислению количества заявок находящихся в накопителе, но с учетом ограничения времени моделирования СМО( по смыслу это означает успеют ли компьютеры за установленное время моделирования СМО обработать все заявки, которые находятся в накопителе или нет ).

Программа состоит из 4 классов :

1. class Clients
2. class Operator
3. class Computer
4. class Modeling

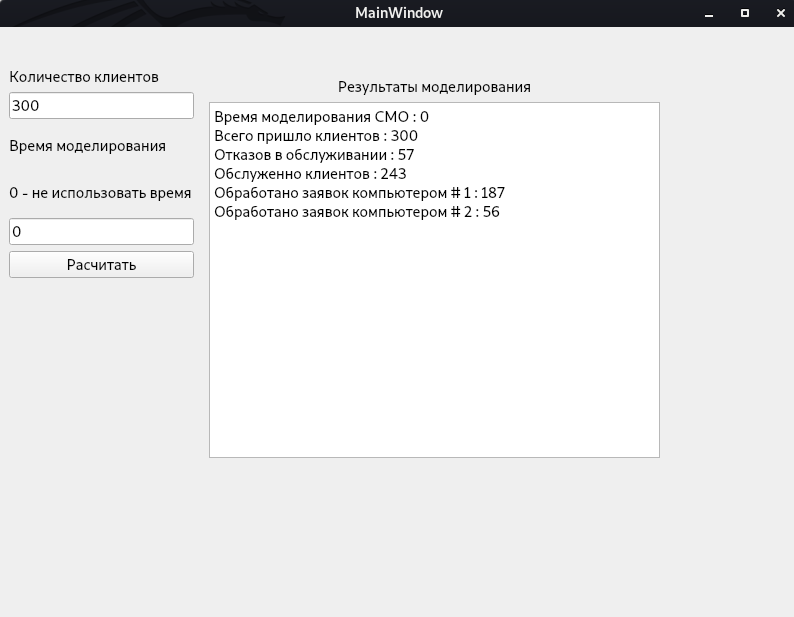
Класс Clients описывает клиентов, класс Operator описывает операторов, класс Computer компьютеры, класс Modeling моделирует СМО.

Для выполнения работы использовалась среда разработки Qt Creator 4.11.0 Based on Qt 5.12.5 ( GCC 9.2.1 20200104, 64 bit ), язык программирования Qt C++( стандарт языка С++17 ). Программа собрана под ОС Linux Debian 5.4.8-1kali1 ( 2020-01-06 ) x86\_64.

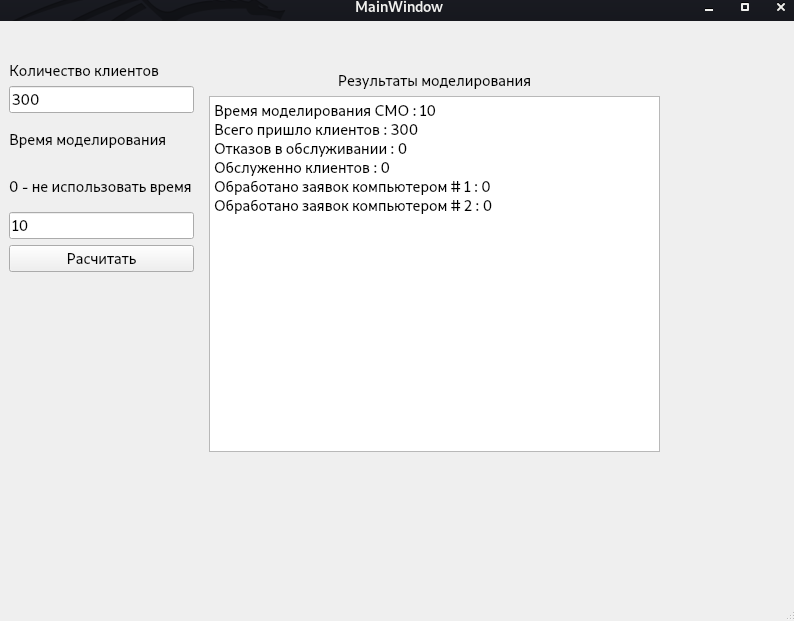
Пользователь вводит количество заявок которые необходимо обработать( что по смыслу означает сколько клиентов пришло ). Задает время моделирования ( если указан 0(ноль), то это означает не использовать время при моделировании СМО ). Нажимает кнопку расчитать.

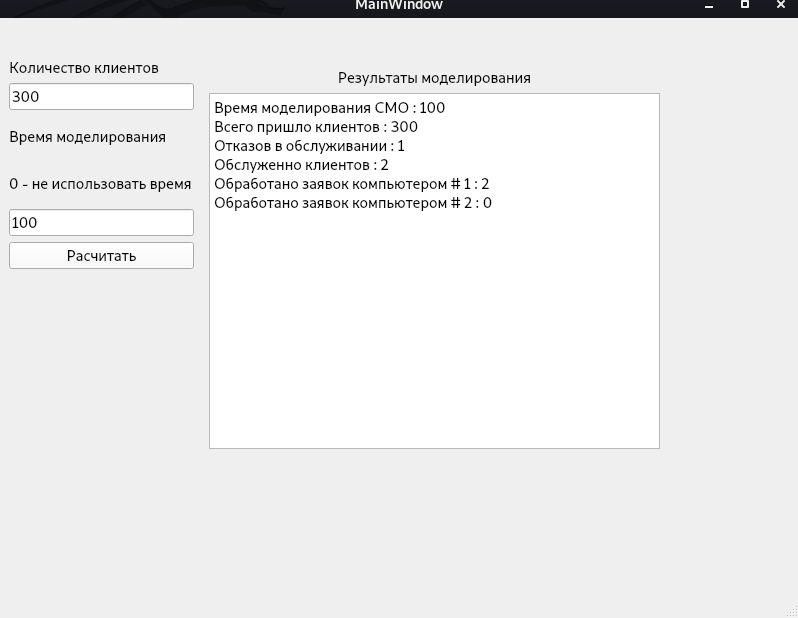
По условию задачи необходимо смоделировать работо СМО для 300 пришедших клиентов.

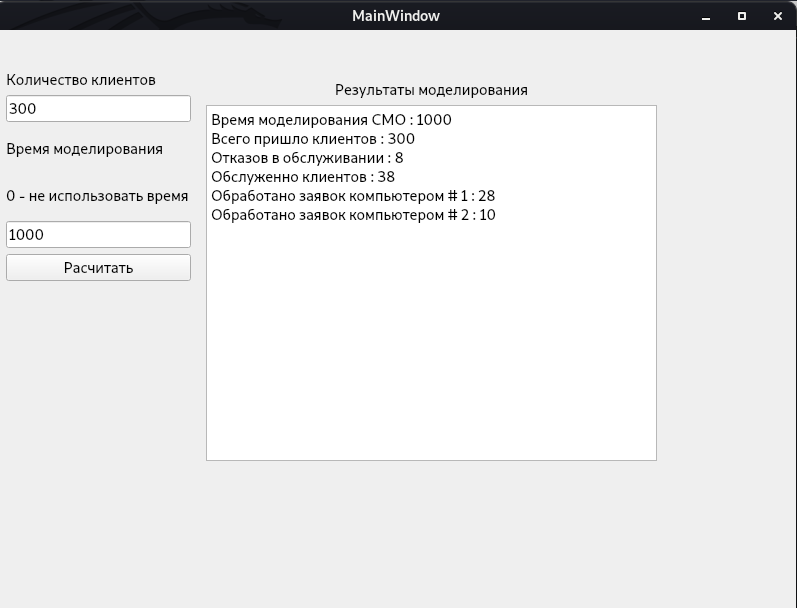
Моделирование 300 заявок без использования времени при моделировании СМО :

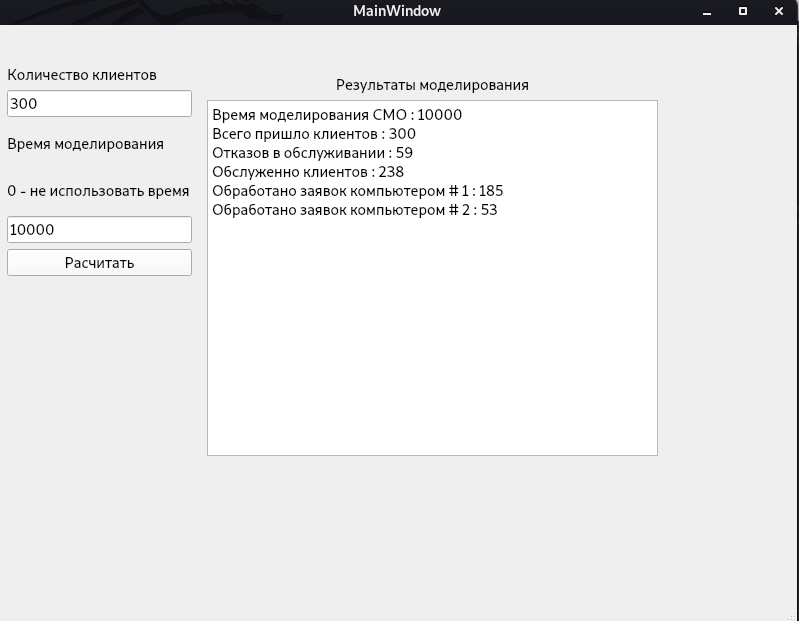


Моделирование 300 заявок с ограничением времени моделирования СМО :









Листинг программы.

**model.h**

#ifndef MODEL\_H

#define MODEL\_H

#include<QString>

#include<QQueue>

#include<QTime>

#include<QMap>

#include<QList>

#include<QPair>

#include<algorithm>

#include<random>

*namespace* MDL {

*class* Clients{

*private* :

QQueue< double > queue ;

*const* double a ;

*const* double b ;

*public* :

Clients( int &size, *const* double &a, *const* double &b )

:a(a),b(b){ CreateClients(size); }

double Generate() {

std::random\_device rd ;

std::mt19937 gen( rd() ) ;

std::uniform\_real\_distribution<> dis( a, b ) ;

*return* dis( gen ) ;

}

void CreateClients( int size ) {

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

queue.push\_back( Generate() ) ;

}

QQueue< double >& GetQueue() { *return* *this*->queue ; }

};

*class* Operator{

*private* :

double current\_task ;

*const* double a ;

*const* double b ;

*public* :

Operator( *const* double &a, *const* double &b )

:current\_task(0), a(a),b(b){ }

double Generate() {

std::random\_device rd ;

std::mt19937 gen( rd() ) ;

std::uniform\_real\_distribution<> dis( a, b ) ;

*return* dis( gen ) ;

}

double GetCurrentTask() *const* { *return* current\_task ; }

void SetCurentTask( double current\_task ) { *this*->current\_task = current\_task; }

};

*class* Computer{

*private*:

*const* double time\_processing\_task = 0 ;

QQueue< double > queue ;

*public*:

Computer( *const* double time\_processing\_task )

:time\_processing\_task(time\_processing\_task) {}

QQueue< double >& GetQueue() { *return* *this*->queue ; }

double Get\_time\_processing\_task() *const* { *return* time\_processing\_task ; }

};

*class* Modeling {

*private* :

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

*struct* Statistics {

*private*:

QMap< QString , uint64\_t > stats ;

*public* :

void Add( *const* QString &s, *const* uint64\_t &i ) { stats[s] = i ; }

uint64\_t GetInfoWithKey( *const* QString &s ) { *return* stats[s] ; }

} STAT ;

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

*public* :

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

Statistics& GetStat() { *return* STAT ; }

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

*public* :

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

Modeling() {}

Modeling( *const* Modeling &\_copy ) = *delete* ;

*const* Modeling& *operator*=( *const* Modeling &modeling ) = *delete* ;

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

*virtual* ~*Modeling*() {}

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

bool isFree( *const* QMap< int, double > &task\_in\_processing ) {

*for*( *auto* it = task\_in\_processing.begin() ; it != task\_in\_processing.end() ; ++it ){

*if*( it.value() == 0 ) *return* *true* ;

}

*return* *false* ;

}

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

int BestCapacity( *const* QMap< int, int > &capacity\_operators,

*const* QMap< int, double > &task\_in\_processing ) {

QSet< QPair< int, double > > list\_task\_in\_processing ;

*for*( *auto* it = task\_in\_processing.begin() ; it != task\_in\_processing.end() ; ++it )

*if*( it.value() == 0 ) list\_task\_in\_processing.insert( qMakePair( it.key(), it.value() ) ) ;

QSet< QPair< int, int > > list\_capacity\_operators ;

*for*( *auto* it = capacity\_operators.begin() ; it != capacity\_operators.end() ; ++it )

list\_capacity\_operators.insert(qMakePair( it.value(), it.key() ) ) ;

*for*( *auto* it\_list\_capacity\_operators = list\_capacity\_operators.rbegin() ;

it\_list\_capacity\_operators != list\_capacity\_operators.rend() ;

++ it\_list\_capacity\_operators )

*for*( *auto* it\_list\_task\_in\_processing = list\_task\_in\_processing.begin() ;

it\_list\_task\_in\_processing != list\_task\_in\_processing.end() ;

++ it\_list\_task\_in\_processing )

*if*( it\_list\_capacity\_operators->second == it\_list\_task\_in\_processing->first )

*return* it\_list\_task\_in\_processing->first ;

*return* -1;

}

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

void Run( double &time\_modeling,

MDL::Clients \*CLIENTS, QVector< MDL::Operator \* > &OPERATORS, QVector< MDL::Computer \* > &COMPUTERS,

QMap< int, double > &task\_in\_processing, QMap< int, int > &capacity\_operators ) {

*const* double step\_time = 0.01 ;

STAT.Add( "time\_modeling", time\_modeling ) ;

double time\_modeling\_for\_computer = time\_modeling ;

*//МОДЕЛИРУЕМ* *РАБОТУ* *ОПЕРАТОРОВ*

STAT.Add( "all\_tasks\_modeling\_stage\_#\_1", *static\_cast*<int>(CLIENTS->GetQueue().size() ) ) ;

double current\_time\_stage\_2 = 0 ;

int count\_rejection\_tasks = 0 ;

int count\_acception\_tasks = 0 ;

double time\_start = 0 ;

double time\_for\_computers = 0 ;

QVector< bool > is\_add\_first\_task = QVector< bool >()={ *false*,*false*,*false* } ;

*if*( time\_modeling > 0 ){

*while*( CLIENTS->GetQueue().size() > 0 && time\_modeling > time\_start ) {

current\_time\_stage\_2 += step\_time ;

time\_start+= step\_time;

time\_modeling -= step\_time ;

*for*( *auto* it = task\_in\_processing.begin() ; it != task\_in\_processing.end() ; it++ ){

*if*( it.value() - step\_time > 0 && is\_add\_first\_task[it.key()] ) it.value() -= step\_time ;

*else* *if*( it.value() - step\_time <= 0 && is\_add\_first\_task[it.key()] ) {

*if*( time\_for\_computers == 0 ) { time\_for\_computers = time\_start; }

is\_add\_first\_task[it.key()] = *false* ;

it.value() = 0 ;

count\_acception\_tasks++ ;

*if*( it.key() == 0 || it.key() == 1 ) {

COMPUTERS[0]->GetQueue().push\_back( OPERATORS[it.key()]->GetCurrentTask()) ;

OPERATORS[it.key()]->SetCurentTask(0);

}

*else* *if*( it.key() == 2 ){

COMPUTERS[1]->GetQueue().push\_back( OPERATORS[it.key()]->GetCurrentTask()) ;

OPERATORS[it.key()]->SetCurentTask(0);

}

}

}

*if*( (CLIENTS->GetQueue().size() > 0 ) && (CLIENTS->GetQueue().head() <= current\_time\_stage\_2 ) ){

*if*( isFree(task\_in\_processing) ) {

*if*(CLIENTS->GetQueue().size() > 0 ) CLIENTS->GetQueue().pop\_front() ;

current\_time\_stage\_2 = 0 ;

int best\_capacity = BestCapacity( capacity\_operators, task\_in\_processing ) ;

*if*( best\_capacity != -1 ) {

double new\_task = OPERATORS[best\_capacity]->Generate();

OPERATORS[best\_capacity]->SetCurentTask( new\_task ) ;

task\_in\_processing[best\_capacity] = new\_task ;

is\_add\_first\_task[best\_capacity] = *true* ;

} *else* *continue* ;

}*else* { count\_rejection\_tasks++ ; *if*( CLIENTS->GetQueue().size() > 0) CLIENTS->GetQueue().pop\_front() ; current\_time\_stage\_2 = 0 ; }

}

}*//while*

STAT.Add( "count\_rejection\_tasks\_modeling\_stage\_#\_1", count\_rejection\_tasks ) ;

STAT.Add( "count\_acception\_tasks\_modeling\_stage\_#\_1", count\_acception\_tasks ) ;

*//МОДЕЛИРУЕМ* *РАБОТУ* *ОПЕРАТОРОВ*

*//МОДЕЛИРУЕМ* *РАБОТУ* *КОМПЬЮТЕРОВ*

*//* *Время* *когда* *была* *обработана* *первая* *заявка* *оператором* *-* *начальное* *время* *для* *обработки* *компьютером*

*//time\_for\_computers;*

time\_modeling\_for\_computer -= time\_for\_computers ;

double cur\_task\_comp\_1 = 0 ;

int64\_t count\_acception\_tasks\_with\_computer\_1 = 0 ;

double cur\_task\_comp\_2 = 0 ;

int64\_t count\_acception\_tasks\_with\_computer\_2 = 0 ;

*while*( COMPUTERS[0]->GetQueue().size() != 0 && (time\_modeling\_for\_computer) > cur\_task\_comp\_1 ) {

*if*( COMPUTERS[0]->GetQueue().size() > 0 ) { cur\_task\_comp\_1 += COMPUTERS[0]->GetQueue().head() ;

COMPUTERS[0]->GetQueue().pop\_front() ;

count\_acception\_tasks\_with\_computer\_1++ ;

}

}

*while*( COMPUTERS[1]->GetQueue().size() != 0 && (time\_modeling\_for\_computer) > cur\_task\_comp\_2 ) {

*if*( COMPUTERS[1]->GetQueue().size() > 0 ) { cur\_task\_comp\_2 += COMPUTERS[1]->GetQueue().head() ;

COMPUTERS[1]->GetQueue().pop\_front() ;

count\_acception\_tasks\_with\_computer\_2++ ;

}

}

STAT.Add( "count\_acception\_tasks\_modeling\_stage\_#\_2\_with\_computer\_1", count\_acception\_tasks\_with\_computer\_1 ) ;

STAT.Add( "count\_acception\_tasks\_modeling\_stage\_#\_2\_with\_computer\_2", count\_acception\_tasks\_with\_computer\_2 ) ;

*//МОДЕЛИРУЕМ* *РАБОТУ* *КОМПЬЮТЕРОВ*

}

*else* *if*( time\_modeling == 0 ) {

*while*( CLIENTS->GetQueue().size() > 0 ) {

qDebug()<< CLIENTS->GetQueue().size() << "\n";

current\_time\_stage\_2 += step\_time ;

time\_start+= step\_time;

*for*( *auto* it = task\_in\_processing.begin() ; it != task\_in\_processing.end() ; it++ ){

*if*( it.value() - step\_time > 0 && is\_add\_first\_task[it.key()] ) it.value() -= step\_time ;

*else* *if*( it.value() - step\_time <= 0 && is\_add\_first\_task[it.key()] ) {

*if*( time\_for\_computers == 0 ) { time\_for\_computers = time\_start; }

is\_add\_first\_task[it.key()] = *false* ;

it.value() = 0 ;

*if*( it.key() == 0 || it.key() == 1 ) {

COMPUTERS[0]->GetQueue().push\_back( OPERATORS[it.key()]->GetCurrentTask()) ;

OPERATORS[it.key()]->SetCurentTask(0);

count\_acception\_tasks++ ;

}

*else* *if*( it.key() == 2 ){

COMPUTERS[1]->GetQueue().push\_back( OPERATORS[it.key()]->GetCurrentTask()) ;

OPERATORS[it.key()]->SetCurentTask(0);

count\_acception\_tasks++ ;

}

}

}

*if*( (CLIENTS->GetQueue().size() != 0 ) && (CLIENTS->GetQueue().head() <= current\_time\_stage\_2 ) ){

*if*( isFree(task\_in\_processing) ) {

*if*(CLIENTS->GetQueue().size() != 0 ) CLIENTS->GetQueue().pop\_front() ;

current\_time\_stage\_2 = 0 ;

int best\_capacity = BestCapacity( capacity\_operators, task\_in\_processing ) ;

*if*( best\_capacity != -1 ) {

double new\_task = OPERATORS[best\_capacity]->Generate();

OPERATORS[best\_capacity]->SetCurentTask( new\_task ) ;

task\_in\_processing[best\_capacity] = new\_task ;

is\_add\_first\_task[best\_capacity] = *true* ;

} *else* *continue* ;

}*else* { count\_rejection\_tasks++ ; *if*( CLIENTS->GetQueue().size() != 0) CLIENTS->GetQueue().pop\_front() ; current\_time\_stage\_2 = 0 ; }

}

*if*( CLIENTS->GetQueue().size() == 0 ) {

*for*( *auto* it = task\_in\_processing.begin() ; it != task\_in\_processing.end() ; it++ ){

*if*( it.key() == 0 || it.key() == 1 ) {

COMPUTERS[0]->GetQueue().push\_back( OPERATORS[it.key()]->GetCurrentTask()) ;

OPERATORS[it.key()]->SetCurentTask(0);

count\_acception\_tasks++ ;

}

*else* *if*( it.key() == 2 ){

COMPUTERS[1]->GetQueue().push\_back( OPERATORS[it.key()]->GetCurrentTask()) ;

OPERATORS[it.key()]->SetCurentTask(0);

count\_acception\_tasks++ ;

}

}

}

}*//while*

STAT.Add( "count\_rejection\_tasks\_modeling\_stage\_#\_1", count\_rejection\_tasks ) ;

STAT.Add( "count\_acception\_tasks\_modeling\_stage\_#\_1", count\_acception\_tasks ) ;

*//МОДЕЛИРУЕМ* *РАБОТУ* *ОПЕРАТОРОВ*

*//МОДЕЛИРУЕМ* *РАБОТУ* *КОМПЬЮТЕРОВ*

STAT.Add( "count\_acception\_tasks\_modeling\_stage\_#\_2\_with\_computer\_1", COMPUTERS[0]->GetQueue().size() ) ;

STAT.Add( "count\_acception\_tasks\_modeling\_stage\_#\_2\_with\_computer\_2", COMPUTERS[1]->GetQueue().size() ) ;

*//МОДЕЛИРУЕМ* *РАБОТУ* *КОМПЬЮТЕРОВ*

}

}

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//*

} ;

*//--------------------------------------------------------------------------------------------------------//*

}

#endif *//* *MODEL\_H*

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

*namespace* Ui { *class* MainWindow; }

QT\_END\_NAMESPACE

*class* MainWindow : *public* QMainWindow

{

Q\_OBJECT

*public*:

MainWindow(QWidget \*parent = *nullptr*);

~*MainWindow*();

*private* slots:

void on\_pushButton\_clicked();

*private*:

Ui::MainWindow \*ui;

};

#endif *//* *MAINWINDOW\_H*

***mainwindow.cpp***

*#include<QMessageBox>*

#include<sstream>

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "model.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

}

MainWindow::~*MainWindow*()

{

*delete* ui;

}

void MainWindow::on\_pushButton\_clicked() {

QString str\_l\_e = ui->lineEdit\_2->text();

*if*( str\_l\_e.isEmpty() ) QMessageBox::information( NULL, QObject::tr( "Информация" ),

tr("Вы не ввели время моделирования, будет использован 0( ноль )") ) ;

int count\_task = ui->lineEdit->text().toInt() ;

*if*( count\_task <= 0 ) QMessageBox::information( NULL, QObject::tr( "Информация" ),

tr("Количество клиентов это положительное число!") ) ;

double time\_modeling = ui->lineEdit\_2->text().toDouble() ;

*if*( time\_modeling < 0 ) QMessageBox::information( NULL, QObject::tr( "Информация" ),

tr("Время должно быть положительное число или 0 если время в модели не учитывается") ) ;

MDL::Modeling \* MODEL = *new* MDL::Modeling();

MDL::Clients \*CLIENTS = *new* MDL::Clients(*count\_task*, 8, 12);

QVector< MDL::Operator \* > OPERATORS ;

OPERATORS.append( *new* MDL::Operator( 15, 25 ) ) ;

OPERATORS.append( *new* MDL::Operator( 30, 50 ) ) ;

OPERATORS.append( *new* MDL::Operator( 20, 60 ) ) ;

QVector< MDL::Computer \* > COMPUTERS ;

COMPUTERS.append( *new* MDL::Computer( 15 ) ) ;

COMPUTERS.append( *new* MDL::Computer( 30 ) ) ;

QMap< int, int > capacity\_operators = QMap< int, int >();

capacity\_operators[0] = 0 ;

capacity\_operators[1] = 0 ;

capacity\_operators[2] = 0 ;

QMap< int, double > task\_in\_processing = QMap< int, double >();

task\_in\_processing[0] = 0 ;

task\_in\_processing[1] = 0 ;

task\_in\_processing[2] = 0 ;

MODEL->Run( time\_modeling, CLIENTS, OPERATORS, COMPUTERS, task\_in\_processing, capacity\_operators ) ;

std::stringstream ss;

QString str0 = "time\_modeling";

ss << "Время моделирования СМО : " << MODEL->GetStat().GetInfoWithKey(str0) << "\n" ;

QString str = "all\_tasks\_modeling\_stage\_#\_1" ;

ss << "Всего пришло клиентов : " << MODEL->GetStat().GetInfoWithKey(str) << "\n" ;

QString str3 = "count\_rejection\_tasks\_modeling\_stage\_#\_1" ;

ss << "Отказов в обслуживании : " << MODEL->GetStat().GetInfoWithKey(str3) << "\n" ;

QString str4 = "count\_acception\_tasks\_modeling\_stage\_#\_1" ;

ss << "Обслуженно клиентов : " << MODEL->GetStat().GetInfoWithKey(str4) << "\n" ;

QString str5 = "count\_acception\_tasks\_modeling\_stage\_#\_2\_with\_computer\_1" ;

ss << "Обработано заявок компьютером # 1 : " << MODEL->GetStat().GetInfoWithKey(str5) << "\n" ;

QString str6 = "count\_acception\_tasks\_modeling\_stage\_#\_2\_with\_computer\_2" ;

ss << "Обработано заявок компьютером # 2 : " << MODEL->GetStat().GetInfoWithKey(str6) << "\n" ;

QString s = QString::fromStdString(ss.str());

ui->textBrowser->setText( s ) ;

*delete* MODEL ;

*delete* CLIENTS ;

}

***main.cpp***

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

*return* a.exec();

}

***mainwindow.ui***

*<?xml version="1.0" encoding="UTF-8"?>*

*<ui* version="4.0"*>*

*<class>*MainWindow*</class>*

*<widget* class="QMainWindow" name="MainWindow"*>*

*<property* name="geometry"*>*

*<rect>*

*<x>*0*</x>*

*<y>*0*</y>*

*<width>*800*</width>*

*<height>*600*</height>*

*</rect>*

*</property>*

*<property* name="windowTitle"*>*

*<string>*MainWindow*</string>*

*</property>*

*<widget* class="QWidget" name="centralwidget"*>*

*<widget* class="QWidget" name="verticalLayoutWidget\_2"*>*

*<property* name="geometry"*>*

*<rect>*

*<x>*10*</x>*

*<y>*40*</y>*

*<width>*185*</width>*

*<height>*211*</height>*

*</rect>*

*</property>*

*<layout* class="QVBoxLayout" name="verticalLayout\_2"*>*

*<item>*

*<layout* class="QVBoxLayout" name="verticalLayout"*>*

*<item>*

*<widget* class="QLabel" name="label"*>*

*<property* name="text"*>*

*<string>*Количество клиентов*</string>*

*</property>*

*</widget>*

*</item>*

*<item>*

*<widget* class="QLineEdit" name="lineEdit"*/>*

*</item>*

*</layout>*

*</item>*

*<item>*

*<widget* class="QLabel" name="label\_3"*>*

*<property* name="text"*>*

*<string>*Время моделирования*</string>*

*</property>*

*</widget>*

*</item>*

*<item>*

*<widget* class="QLabel" name="label\_4"*>*

*<property* name="text"*>*

*<string>*0 - не использовать время*</string>*

*</property>*

*</widget>*

*</item>*

*<item>*

*<widget* class="QLineEdit" name="lineEdit\_2"*/>*

*</item>*

*<item>*

*<widget* class="QPushButton" name="pushButton"*>*

*<property* name="text"*>*

*<string>*Расчитать*</string>*

*</property>*

*</widget>*

*</item>*

*</layout>*

*</widget>*

*<widget* class="QWidget" name="verticalLayoutWidget\_3"*>*

*<property* name="geometry"*>*

*<rect>*

*<x>*210*</x>*

*<y>*50*</y>*

*<width>*451*</width>*

*<height>*381*</height>*

*</rect>*

*</property>*

*<layout* class="QVBoxLayout" name="verticalLayout\_3"*>*

*<item>*

*<widget* class="QLabel" name="label\_2"*>*

*<property* name="text"*>*

*<string>*Результаты моделирования*</string>*

*</property>*

*<property* name="alignment"*>*

*<set>*Qt::AlignCenter*</set>*

*</property>*

*</widget>*

*</item>*

*<item>*

*<widget* class="QTextBrowser" name="textBrowser"*/>*

*</item>*

*</layout>*

*</widget>*

*</widget>*

*<widget* class="QMenuBar" name="menubar"*>*

*<property* name="geometry"*>*

*<rect>*

*<x>*0*</x>*

*<y>*0*</y>*

*<width>*800*</width>*

*<height>*24*</height>*

*</rect>*

*</property>*

*</widget>*

*<widget* class="QStatusBar" name="statusbar"*/>*

*</widget>*

*<resources/>*

*<connections/>*

*</ui>*