

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ  
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,  
МЕХАНИКИ И ОПТИКИ**

Факультет \_\_\_\_\_ компьютерных технологий и управления \_\_\_\_\_  
Кафедра \_\_\_\_\_ вычислительной техники \_\_\_\_\_  
Направление подготовки (специальность) \_\_\_\_\_ 230100 \_\_\_\_\_

**О Т Ч Е Т  
по практике**

Тема задания: \_\_\_\_\_ тема \_\_\_\_\_

Студенты \_\_\_\_\_ Вакс Алексей, группа 3103 \_\_\_\_\_

Руководитель практики \_\_\_\_\_ Соснин В.В. \_\_\_\_\_

Оценка руководителя \_\_\_\_\_

Дата \_\_\_\_\_

Санкт-Петербург  
2013г.

# 1 TeX(LaTeX)

TeX - система компьютерной вёрстки, разработанная Дональдом Кнутom в целях создания компьютерной типографии. В отличие от визуальных редакторов, TeX сам форматирует документ на основе выбранного пользователем шаблона — как правило, хранящегося в файле формата .tex. Далее файлы .tex транслируются в файлы .dvi, которые уже могут быть напечатаны либо перегнаны в формат pdf.

TeX/ предназначен для набора научного текста, на большую часть состоящего из формул, и обеспечивает удобное форматирование и оформление текста, перекрестных ссылок и библиографии, а также позволяет создавать более качественные сложные документы, чем WYSIWYG-редакторы — такие как MS Word.

Документы набираются на собственном низкоуровневом языке разметки TeX, содержащем команды отступа и смены шрифта, а также расширением TeX'a — LaTeX'ом, который содержит в себе набор готовых стилей и шаблонов, позволяющих, например, быстро вставлять оглавление, не задумываясь о нумерации отдельных объектов, что в MS Word гораздо сложнее.

Главным преимуществом TeX является возможность автоматизировать форматирование и структурирование документа, а также более гибкие возможности набора сложных формул, а также обеспечение кроссплатформенности и совместимости с PS и PDF.

Главным недостатком TeX является сложность в освоении и адаптации после WYSIWYG-редакторов, а также некоторые затруднения при работе с графикой, так как необходимо создавать для этого .eps-файл.

## 1.1 MiKTeX

MiKTeX - открытый дистрибутив  $\text{\TeX}$  для платформы Windows, имеющий возможность автоматического обновления установленных компонентов и пакетов.

В состав MiKTeX включены:

- классический  $\text{\TeX}$ -компилятор;
- различные варианты  $\text{\TeX}$ : pdf $\text{\TeX}$ , e- $\text{\TeX}$ , pdf-e- $\text{\TeX}$ , и т.д.;
- конверторы  $\text{\TeX}$  в PDF;
- MetaPost — интерпретатор для графических иллюстраций;
- полный набор общеиспользуемых макропакетов: LaTeX, ConTeXt и др.;
- средство просмотра Yip;
- инструменты и утилиты;

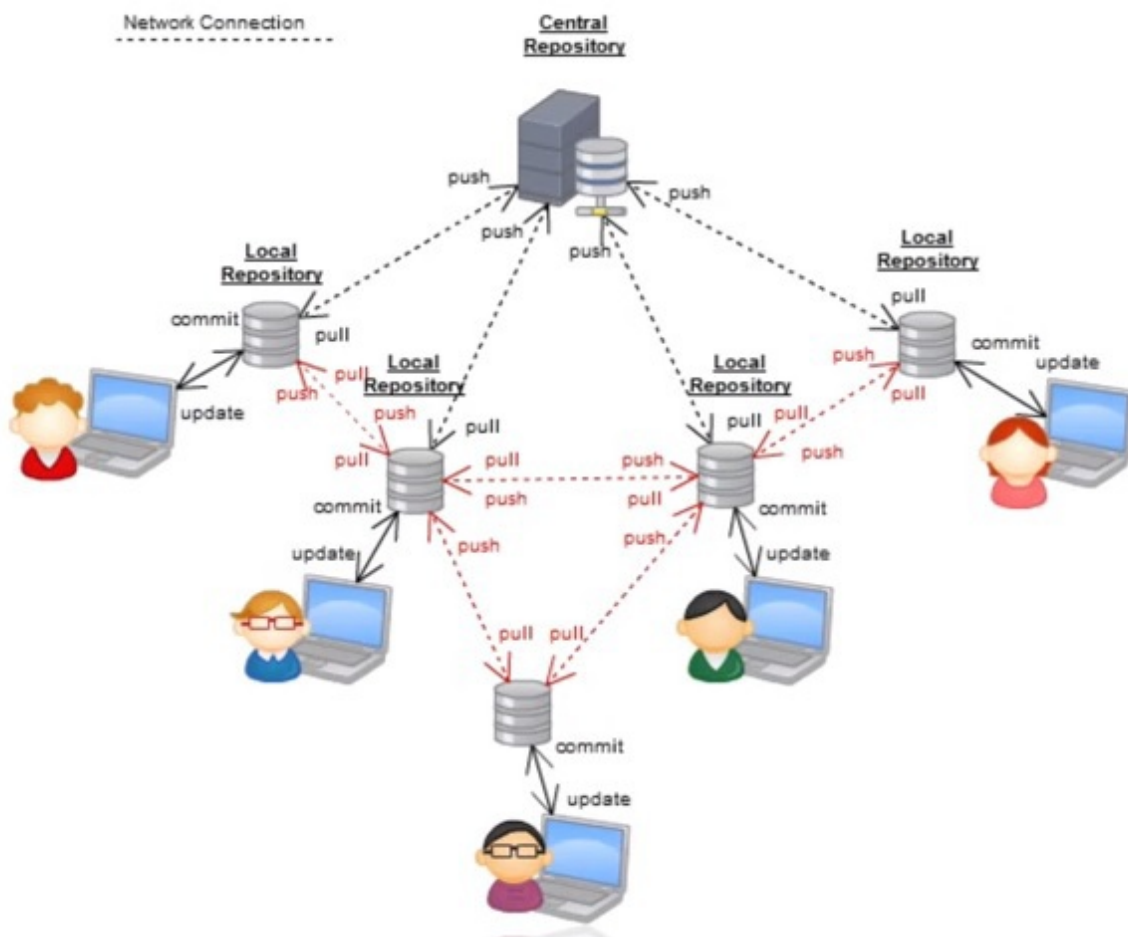
## 2 Git

Git - система управления версиями файлов, предназначенная для хранения, разделения и слияния версий и сохранения всей истории разработки в репозитории.

Для сохранения изменений используется команда `commit`.

Система контроля версий необходима для того, чтобы каждый из группы разработчиков видел свои собственные изменения, а также изменения, сделанные его коллегами, чтобы организовать быструю замену нужных файлов для замены всех модулей на модули последних версий, а также для возможностей возврата к предыдущим.

В Git между главным репозиторием и пользователем существует промежуточный репозиторий — локальный, который обеспечивает работу системы без интернета.

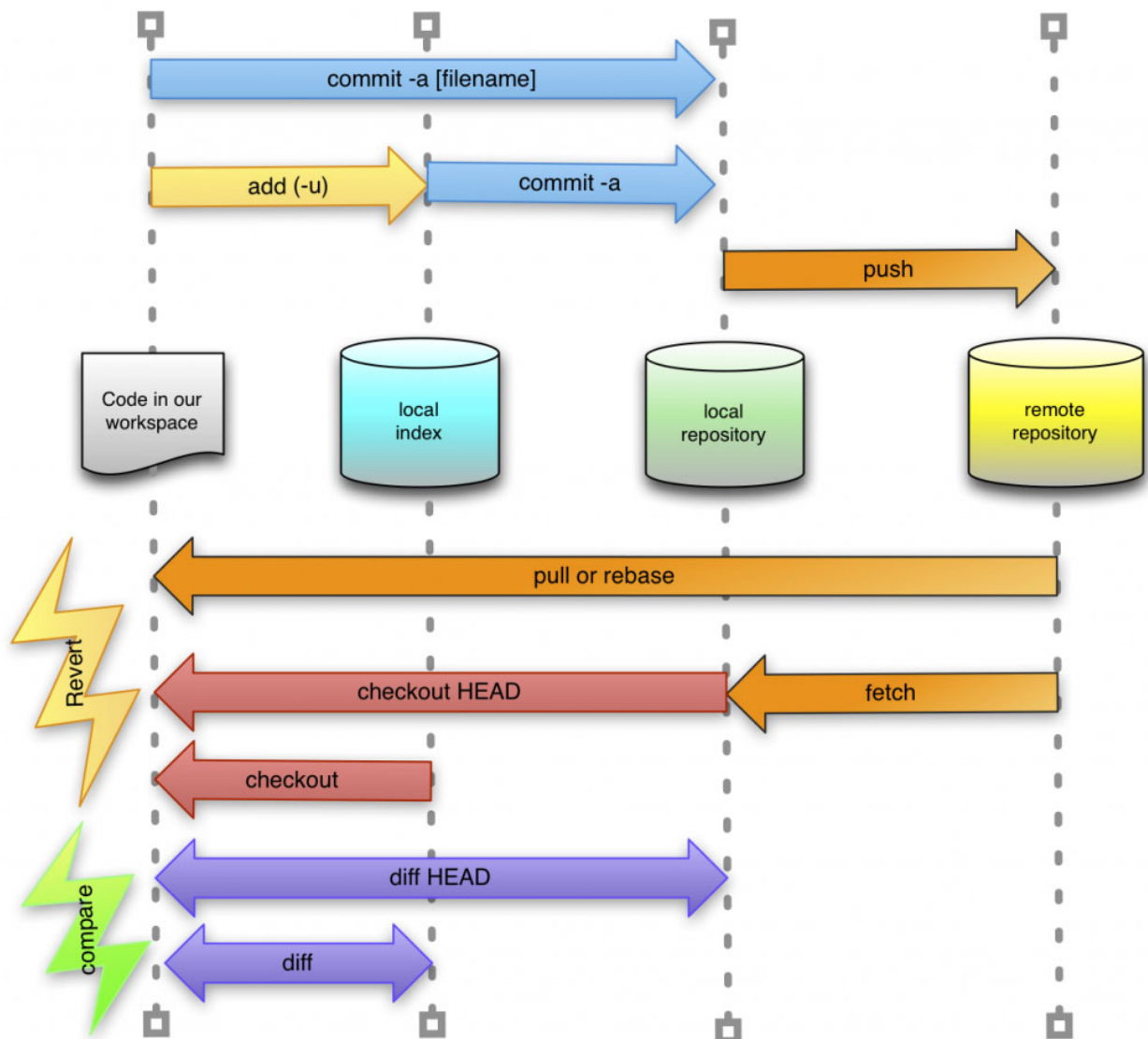


## 2.1 Описание работы Git:

Сотрудник работает в своем рабочем пространстве (workspace), система параллельно с его изменениями вносит изменения в главный индекс, следя за теми файлами, которые добавил сотрудник. После этого сотрудник может сохранить текущее состояние. Далее файлы добавляются в локальный репозиторий.

При наличии интернета сотрудник может перенести файлы в удаленный репозиторий (например, github). После этого из удаленного репозитория можно загрузить все файлы с текущими изменениями, а также просмотреть

реть изменения текущей версии. Кроме того, другие пользователи также могут одновременно загружать свои изменения.



## 2.2 Основные git-команды:

**git init** - создание нового каталога с файлами нового репозитория.

**git commit** - сохранение изменений.

**git status** - вывод информации обо всех изменениях, внесенных в дерево директорий проекта по сравнению с последним коммитом рабочей ветки.

**git add** - ввод в индекс (временное хранилище) изменений, которые затем войдут в commit.

*Основные ключи, используемые для упрощения работы с git:*

**git commit -a** - выполняет commit, индексируя изменения в файлах проекта без индексации новых файлов, но с учетом удаления.

**git commit -m «commit comment»** - комментируем commit из командной строки вместо текстового редактора.

**git commit "filename"** - вносит в индекс и создаёт commit на основе изменений только одного файла.

**git reset** - возврат к определенному commit'у, откат изменений, «жесткий» или «мягкий». Мягкий оставляет индекс и дерево файлов и директорий нетронутым, а жёсткий возвращает указанное состояние, удаляя все последующие изменения, вызванные commit'ами.

**git diff** - просмотр изменений, не внесенных в индекс.

**git push** - внесение изменения в удаленный репозиторий.

**git pull** - возвращает изменения из удаленного репозитория.

Использованные материалы: <http://git-scm.com/>, <http://robotics.usc.edu/>.

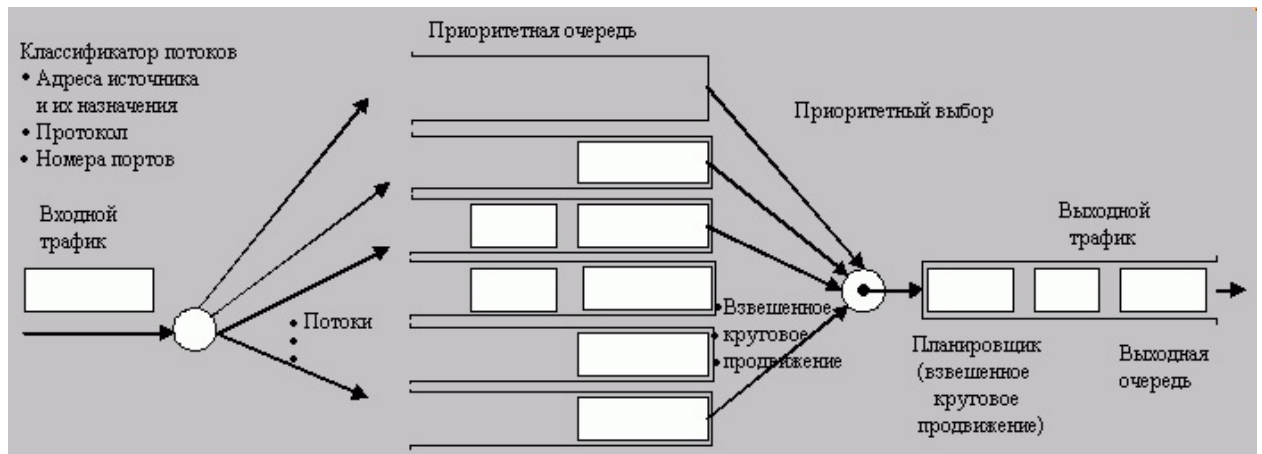
### 3 Взвешенная справедливая очередь

Взвешенная справедливая очередь (англ. Weighted fair queuing, WFQ) — механизм планирования пакетных потоков данных с различными приоритетами. Его целью является регулирование использования одного канала передачи данных несколькими конкурирующими очередями пакетов данных.

Взвешенная справедливая очередь — это комбинированный механизм, сочетающий приоритетное обслуживание с взвешенным и используя принципы и того и другого. От приоритетного обслуживания взята возможность устанавливать предпочтение одним очередям пакетов данных, а от взвешенного обслуживания взята возможность предоставлять всем очере-

для пакетов заявок определенный минимум пропускной способности, чтобы гарантировать некоторые требования к величине задержек.

Типичный вариант реализации WFQ, когда всем классам трафика достаются равные доли пропускной способности выходного интерфейса из оставшейся от трафика приоритетного класса доли:



### 3.1 Реализация WFQ на языке программирования

Напишем программу, эмулирующую работу системы массового обслуживания по механизму WFQ с экспоненциальным распределением размеров пакетов (т.е. длительности обслуживания) и экспоненциальным потоком пакетов на языке C-Sharp в консольном приложении.

Listing 1: Source Code

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading;
6
7 namespace ConsoleApplication2
8 {
9     class Program
10    {

```

```

11 public static double Gen(double mat) //генератор случайной
    величины, распределенной по экспоненциальному закону
12 {
13     double x = 0;
14     var rnd = new Random();
15     while (x == 0) x = -mat * (Math.Log(rnd.NextDouble()));
16     return x;
17 }
18 public static bool ifnotnull(double[] arr)
19 {
20     bool ans = false;
21     foreach (double x in arr)
22         if (x != 0) ans = true; //значит, очередь не пуста
23     return ans;
24 }
25 public static bool ifnotfull(double[] arr)
26 {
27     bool ans = false;
28     foreach (double x in arr)
29         if (x == 0)
30         {
31             ans = true; //значит, в очереди есть свободное
                место
32             break;
33         }
34     return ans;
35 }
36 public static int numlast(double[] arr)
37 {
38     int ans = arr.Length;
39     for (int i = 0; i < arr.Length; i++)
40         if (arr[i] == 0)
41         {
42             ans = i; //возврат номера последнего свободного
                места в очереди
43             break;

```



```

44         }
45         return ans;
46     }
47     static void Main(string[] args)
48     {
49         double timer = 0.08; //минимальная единица времени
50         double timewait1sum = 0;
51         int timewait1kol = 0;
52         double timewait2sum = 0;
53         int timewait2kol = 0;
54         double finaltimewait1 = 0;
55         double finaltimewait2 = 0;
56         //переменные для определения среднего времени ожидания для
           разных накопителей
57         double buf1m = 0, buf2m = 0; //расчет средней длины очереди
58         double buf1ms = 0, buf2ms = 0;
59         int kol = 0;
60         int buf1size = 280, buf2size = 180; //размеры очередей
61         int buf1 = 0, buf2 = 0;
62         double[] arr1 = new double[buf1size]; //массивы с размерами
           пакетов (длительностью обслуживания заявок)
63         double[] arr2 = new double[buf2size];
64         double servtime1 = 0.2, servtime2 = 0.71; //среднее время
           обслуживания
65         double intensity1 = 4.1, intensity2 = 2.2; //средняя
           интенсивность
66         double kpd = 0;
67         double load = 0;
68         //гибкое переключение на другую очередь после обработки
           определенного кол-ва заявок, зависит от величины [
           Количество_недавно_обслуженных_заявок*Их_вес(вес равен
           длительности обслуживания)]
69         double SWITCHINGFINAL1 = 40; // переключение после
           обработки столькох заявок 1й очереди
70         double SWITCHINGFINAL2 = 60; // переключение после
           обработки столькох заявок 2й очереди

```

```

71 double SWITCHINGPROGRESS = 0;
72 //расширим эту переменную для переключения по величине,
   зависящей от конкретной очереди
73 double SWITCHING = 1;
74 //обнуление массивов
75 for (int i = 0; i < buf1size; i++)
76 {
77     arr1[i] = 0;
78 }
79 for (int i = 0; i < buf2size; i++)
80 {
81     arr2[i] = 0;
82 }
83 //
84 //Главный цикл работы системы:
85 //
86 for (double i = timer; i < 100000; i += timer)
87 {
88     //добавляем новые заявки в очередь
89     kol = Convert.ToInt32(Gen(timer * intensity1));
90     for (int j = 0; j < kol; j++)
91     {
92         if (ifnotfull(arr1))
93         {
94             arr1[numlast(arr1)] = Gen(servtime1);
95             timewait1kol++;
96         }
97     }
98     kol = Convert.ToInt32(Gen(timer * intensity2));
99     for (int j = 0; j < kol; j++)
100     {
101         if (ifnotfull(arr2))
102         {
103             arr2[numlast(arr2)] = Gen(servtime2);
104             timewait2kol++;
105         }

```

```

106     }
107     //закончили добавлять новые заявки , теперь обслуживаем
108     заявки , проверка переключения
109
110     if ((SWITCHINGPROGRESS >= SWITCHINGFINAL1) && (
111         SWITCHING == 1))
112     {
113         SWITCHING = 2;
114
115         SWITCHINGPROGRESS = 0; //сбросили прогресс ,
116         переключились на очередь 2
117     }
118
119     if ((SWITCHINGPROGRESS >= SWITCHINGFINAL2) && (
120         SWITCHING == 2))
121     {
122         SWITCHING = 1;
123
124         SWITCHINGPROGRESS = 0; //сбросили прогресс ,
125         переключились на очередь 1
126     }
127
128     //обслуживание заявок
129     if (SWITCHING == 1)
130     {
131         if ((arr1[0] > 0))
132         {
133             arr1[0] -= servtime1;
134             SWITCHINGPROGRESS += servtime1;
135             timewait1sum += servtime1; //прирост времени
136             ожидания
137             if ((arr2[0] > 0))
138             {
139                 timewait2sum += servtime1; //прирост
140                 времени ожидания
141             }
142         }
143     }

```

```

135         }
136         else SWITCHING = 2; //если заявок в первой очереди
           нет, идем во вторую
137     }
138
139     if (SWITCHING == 2)
140     {
141         if ((arr2[0] > 0))
142         {
143             arr2[0] -= servtime2;
144             SWITCHINGPROGRESS += servtime2;
145             timewait2sum += servtime2; //прирост времени
               ожидания
146             if ((arr1[0] > 0))
147             {
148                 timewait1sum += servtime2; //прирост времени
                   ожидания
149             }
150         }
151         else SWITCHING = 1; //если заявок во второй очереди
           нет, идем в первую
152     }
153
154     //закончили обслуживать заявки, теперь сдвигаем массив,
       чтобы на 1м месте не было "пустого места", а была
       заявка
155
156     if ((arr1[0] <= 0))
157     {
158         for (int j = 0; j < buf1size - 1; j++)
159         {
160             arr1[j] = arr1[j + 1];
161         }
162         arr1[buf1size - 1] = 0;
163     }
164

```

```

165     if ((arr2[0] <= 0))
166     {
167         for (int j = 0; j < buf2size - 1; j++)
168         {
169             arr2[j] = arr2[j + 1];
170         }
171         arr2[buf2size - 1] = 0;
172     }
173
174     //подготавливаем информацию для вывода
175     buf1 = numlast(arr1);
176     buf2 = numlast(arr2);
177     buf1ms += buf1;
178     buf2ms += buf2;
179     buf1m = buf1ms / (i / timer);
180     buf2m = buf2ms / (i / timer);
181     if ((timewait1kol != 0) && (timewait2kol != 0))
182     {
183         finaltimewait1 = timewait1sum / timewait1kol;
184         finaltimewait2 = timewait2sum / timewait2kol;
185     }
186     Console.Clear();
187     Console.WriteLine("В настоящее время обрабатываю заявки
        из очереди: " + SWITCHING + "\t");
188     Console.WriteLine("Первая очередь: " + buf1 + "/" +
        buf1size + ". Вторая очередь: " + buf2 + "/" +
        buf2size + "\t");
189     Console.WriteLine("Средняя первая оч.: " + buf1m + ".
        Средняя вторая оч.: " + buf2m + "\t");
190     Console.WriteLine("Среднее время ожидания в 1й оч.: " +
        finaltimewait1 + ". Во 2й оч.: " + finaltimewait2 +
        "\t");
191     Console.WriteLine("Средн. время обл. в 1й очереди: " +
        servtime1 + ". Во 2й очереди: " + servtime2 + "\t")
        ;
192     Console.WriteLine("Средн. интенсивн. заявок 1й оч.: " +

```

```

193         intensity1 + ". Во 2й очереди: " + intensity2 + "\t
        ");
194     Console.WriteLine("Суммарный размер обработ. заявок для
        переключения в 1ю/2ю очередь: " + SWITCHINGFINAL2 +
        "/" + SWITCHINGFINAL1 + "\t");
195     Thread.Sleep(10); //для нормальной отрисовки данных
196 }
197 Console.ReadLine();
198 }
199 }
```

## 4 Выводы

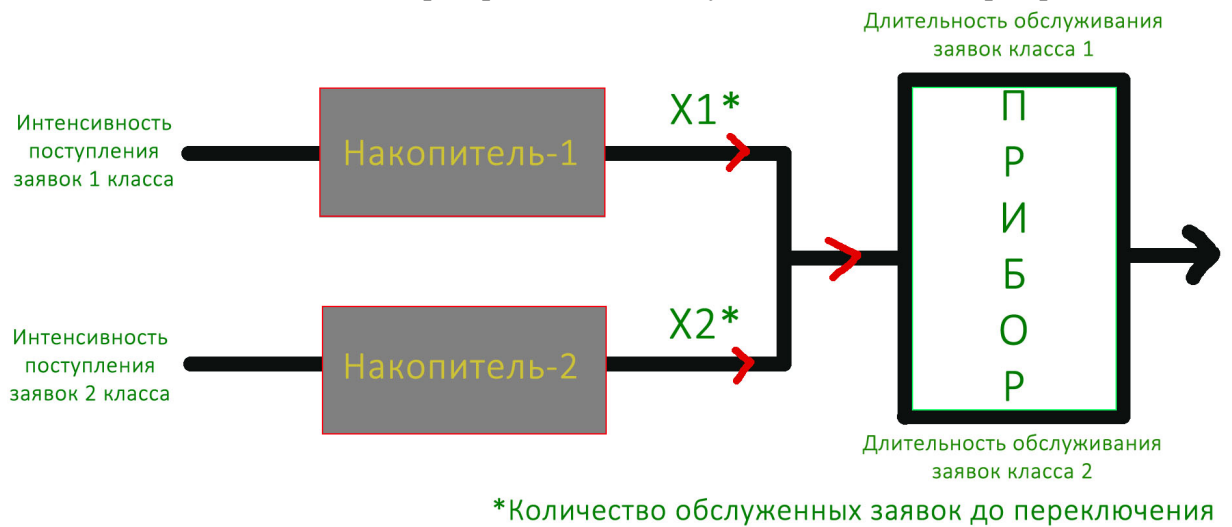
Таким образом, мы написали программу для эмуляции системы массового обслуживания с приоритетом WFQ (Взвешенная справедливая очередь, Weighted fair queuing) с экспоненциальным распределением длительности обслуживания заявок и экспоненциальным потоком заявок для двух классов заявок в двух накопителях. С помощью этой программы можно исследовать зависимости времени ожидания, а также средних и текущих длин очередей для обоих классов заявок от времени и от входных параметров, которые можно удобно менять на другие желаемые.

Технология WFQ позволяет сочетать преимущества взвешенного и приоритетного обслуживания одновременно, позволяя как давать более важным классам заявок повышенный приоритет, так и поддерживать пропускную способность в низших классах заявок на определенном уровне, таким образом, оптимизируя процесс обслуживания заявок (или обработки пакетов) в справедливом (выгодном) порядке.

Приоритет WFQ – один из инструментов обеспечения заданного уровня качества обслуживания в сетях, в связи с чем часто используется в марш-

рутизаторах и коммутаторах.

### Реализация СМО с приоритетом WFQ в написанной программе



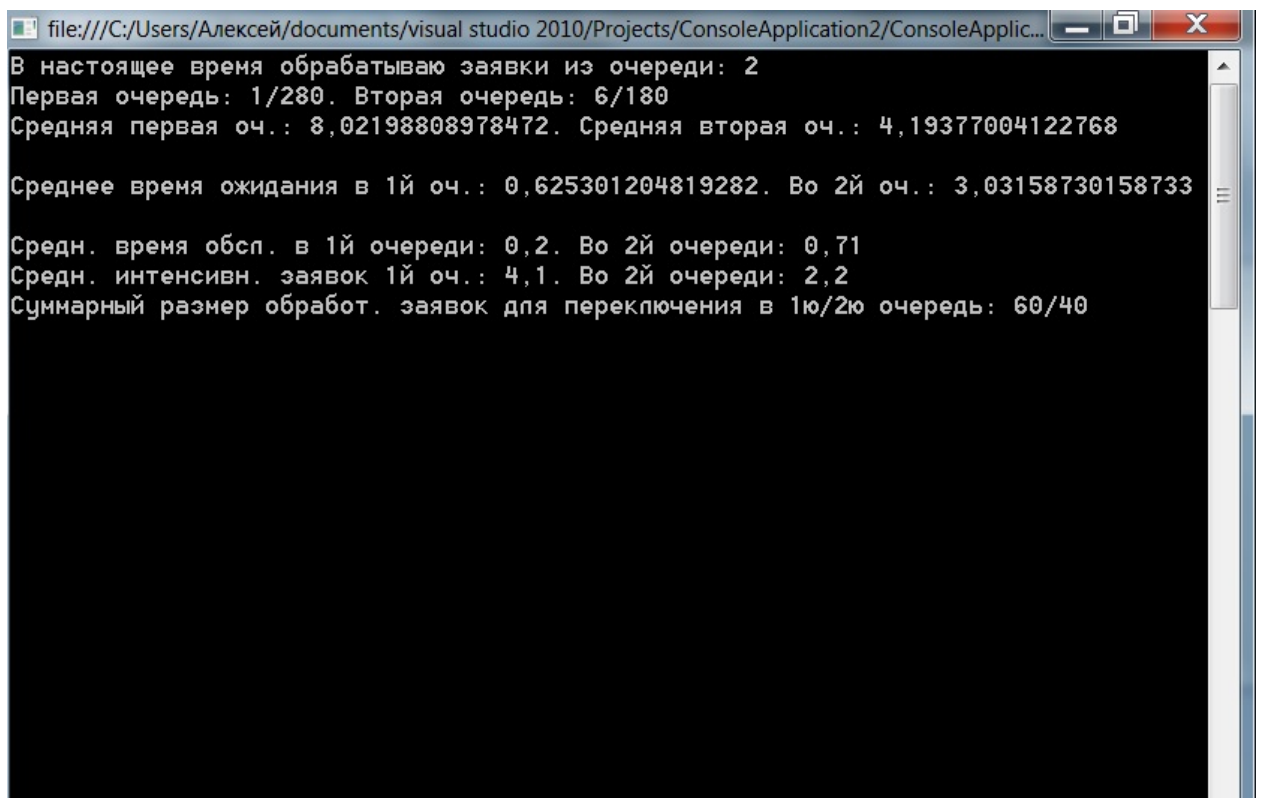
### Результаты работы программы:

```
file:///C:/Users/Алексей/documents/visual studio 2010/Projects/ConsoleApplication2/ConsoleApplic...
В настоящее время обрабатываю заявки из очереди: 1
Первая очередь: 14/280. Вторая очередь: 9/180
Средняя первая оч.: 7,04087736789586. Средняя вторая оч.: 3,59720837487515

Среднее время ожидания в 1й оч.: 0,618075801749248. Во 2й оч.: 2,95313186813184

Средн. время обсп. в 1й очереди: 0,2. Во 2й очереди: 0,71
Средн. интенсивн. заявок 1й оч.: 4,1. Во 2й очереди: 2,2
Суммарный размер обработ. заявок для переключения в 1ю/2ю очередь: 60/40
```

Обслуживание заявок из первого накопителя.



The image shows a screenshot of a console application window. The title bar indicates the file path: file:///C:/Users/Алексей/documents/visual studio 2010/Projects/ConsoleApplication2/ConsoleApplic... The console output displays the following statistics:

```
В настоящее время обрабатываю заявки из очереди: 2
Первая очередь: 1/280. Вторая очередь: 6/180
Средняя первая оч.: 8,02198808978472. Средняя вторая оч.: 4,19377004122768

Среднее время ожидания в 1й оч.: 0,625301204819282. Во 2й оч.: 3,03158730158733

Средн. время обсл. в 1й очереди: 0,2. Во 2й очереди: 0,71
Средн. интенсивн. заявок 1й оч.: 4,1. Во 2й очереди: 2,2
Суммарный размер обработ. заявок для переключения в 1ю/2ю очередь: 60/40
```

Обслуживание заявок из второго накопителя.