**ОТЧЕТ**

по лабораторной работе №4

Разработка и реализация моделирующих алгоритмов систем массового обслуживания

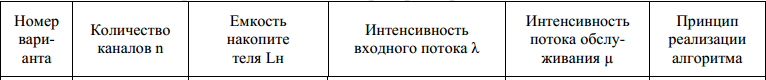
Выполнил студент группы АСОИ-122

Силков Е.В.

Проверил: Кушнер А.В.

***Цель работы:*** изучение принципов построения и реализация  
моделирующих алгоритмов одноканальной СМО; проведение однофакторного  
расчетного эксперимента СМО с помощью разработанной программы

***Исходные данные***



***Код программы***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Threading;

namespace IMS\_01

{

/// <summary>

/// Логика взаимодействия для Lab\_4.xaml

/// </summary>

public partial class Lab\_4 : Page

{

DispatcherTimer requestForming = new DispatcherTimer(),

requestProcessing1 = new DispatcherTimer(),

requestProcessing2 = new DispatcherTimer(),

requestProcessing3 = new DispatcherTimer(),

timeRunningOut = new DispatcherTimer();

int requestsCount, requestsRealCount,

requestsByChannel1, requestsByChannel2, requestsByChannel3;

public const double LAMBDA = 0.44;

public const double MU = 0.4 / 3;

public const int STORAGE\_DEVICE\_LIMIT = int.MaxValue;

//public const int STORAGE\_DEVICE\_LIMIT = 3;

public class Request // Запрос

{

public double Dti { get; } // Интервал времени между поступлениями двух соседних заявок

public double Dtki { get; } // Время обслуживания заявки

public Request()

{

Random r = new Random();

Dti = (-1.0 / LAMBDA) \* Math.Log(r.NextDouble());

Dtki = (-1.0 / MU) \* Math.Log(r.NextDouble());

}

}

Queue<Request> source = new Queue<Request>(), storageDevice = new Queue<Request>(),

completedRequests = new Queue<Request>(), failedRequests = new Queue<Request>();

Request channel1, channel2, channel3;

Request someRequest;

public Lab\_4()

{

InitializeComponent();

requestForming.Tick += RequestForming\_Tick;

requestForming.Interval = new TimeSpan(0, 0, 0, 0, 100);

requestProcessing1.Tick += RequestProcessing1\_Tick;

requestProcessing1.Interval = new TimeSpan(0, 0, 0, 0, 100);

requestProcessing2.Tick += RequestProcessing2\_Tick;

requestProcessing2.Interval = new TimeSpan(0, 0, 0, 0, 100);

requestProcessing3.Tick += RequestProcessing3\_Tick;

requestProcessing3.Interval = new TimeSpan(0, 0, 0, 0, 100);

timeRunningOut.Tick += TimeRunningOut\_Tick;

timeRunningOut.Interval = new TimeSpan(0, 0, 0, 0, 100);

}

private void StartButton\_Click(object sender, RoutedEventArgs e)

{

progressBarT.Maximum = Convert.ToDouble(timeTextBox.Text);

requestsCount = Convert.ToInt32(countTextBox.Text);

someRequest = new Request();

progressBarS.Maximum = someRequest.Dti \* 1000;

Update();

timeRunningOut.Start();

requestForming.Start();

}

private void TimeRunningOut\_Tick(object sender, EventArgs e)

{

progressBarT.Value += 100;

if (progressBarT.Value >= progressBarT.Maximum)

{

progressBarT.Value -= progressBarT.Maximum;

Update();

}

if (completedRequests.Count + failedRequests.Count == Convert.ToInt32(countTextBox.Text))

{

timeRunningOut.Stop();

MessageBox.Show("Вероятность отказа: " + 100 \* failedRequests.Count / requestsRealCount +

"%\nВероятность обслуживания: " + 100 \* completedRequests.Count / requestsRealCount + "%");

}

}

private void RequestProcessing3\_Tick(object sender, EventArgs e)

{

progressBar3.Value += 100;

}

private void RequestProcessing2\_Tick(object sender, EventArgs e)

{

progressBar2.Value += 100;

}

private void RequestProcessing1\_Tick(object sender, EventArgs e)

{

progressBar1.Value += 100;

}

private void RequestForming\_Tick(object sender, EventArgs e)

{

if (requestsRealCount + source.Count == Convert.ToInt32(countTextBox.Text))

{

requestForming.Stop();

progressBarS.Value = 0;

return;

}

progressBarS.Value += 100;

if (progressBarS.Value >= progressBarS.Maximum)

{

progressBarS.Value -= progressBarS.Maximum;

source.Enqueue(someRequest);

someRequest = new Request();

progressBarS.Maximum = someRequest.Dti \* 1000;

requestsLeftLabel.Content = source.Count.ToString();

}

}

public void Update()

{

// переводим заявку из первого канала к завершенным, если такая в нем имеется и она уже обработана

if (channel1 != null && progressBar1.Value >= progressBar1.Maximum)

{

completedRequests.Enqueue(channel1);

progressBar1.Value = 0;

requestProcessing1.Stop();

channel1 = null;

requestsByChannel1++;

channel1Pic.Fill = new SolidColorBrush(Colors.White);

}

// переводим заявку из второго канала к завершенным, если такая в нем имеется и она уже обработана

if (channel2 != null && progressBar2.Value >= progressBar2.Maximum)

{

completedRequests.Enqueue(channel2);

progressBar2.Value = 0;

requestProcessing2.Stop();

channel2 = null;

requestsByChannel2++;

channel2Pic.Fill = new SolidColorBrush(Colors.White);

}

// переводим заявку из третьего канала к завершенным, если такая в нем имеется и она уже обработана

if (channel3 != null && progressBar3.Value >= progressBar3.Maximum)

{

completedRequests.Enqueue(channel3);

progressBar3.Value = 0;

requestProcessing3.Stop();

channel3 = null;

requestsByChannel3++;

channel3Pic.Fill = new SolidColorBrush(Colors.White);

}

// если 1-й канал свободен, пытаемся заполнить его заявкой...

if (channel1 == null)

{

// ... из накопителя

if (storageDevice.Count > 0)

{

channel1 = storageDevice.Dequeue();

if (storageDevice.Count < STORAGE\_DEVICE\_LIMIT) storageDevicePic.Fill = new SolidColorBrush(Colors.White);

progressBar1.Maximum = channel1.Dtki \* 1000;

requestProcessing1.Start();

channel1Pic.Fill = new SolidColorBrush(Colors.Yellow);

}

// ... из источника

else if (source.Count > 0)

{

channel1 = source.Dequeue();

progressBar1.Maximum = channel1.Dtki \* 1000;

requestProcessing1.Start();

channel1Pic.Fill = new SolidColorBrush(Colors.Yellow);

requestsRealCount++;

}

}

// если 2-й канал свободен, пытаемся заполнить его заявкой...

if (channel2 == null)

{

// ... из накопителя

if (storageDevice.Count > 0)

{

channel2 = storageDevice.Dequeue();

if (storageDevice.Count < STORAGE\_DEVICE\_LIMIT) storageDevicePic.Fill = new SolidColorBrush(Colors.White);

progressBar2.Maximum = channel2.Dtki \* 1000;

requestProcessing2.Start();

channel2Pic.Fill = new SolidColorBrush(Colors.Yellow);

}

// ... из источника

else if (source.Count > 0)

{

channel2 = source.Dequeue();

progressBar2.Maximum = channel2.Dtki \* 1000;

requestProcessing2.Start();

channel2Pic.Fill = new SolidColorBrush(Colors.Yellow);

requestsRealCount++;

}

}

// если 3-й канал свободен, пытаемся заполнить его заявкой...

if (channel3 == null)

{

// ... из накопителя

if (storageDevice.Count > 0)

{

channel3 = storageDevice.Dequeue();

if (storageDevice.Count < STORAGE\_DEVICE\_LIMIT) storageDevicePic.Fill = new SolidColorBrush(Colors.White);

progressBar3.Maximum = channel3.Dtki \* 1000;

requestProcessing3.Start();

channel3Pic.Fill = new SolidColorBrush(Colors.Yellow);

}

// ... из источника

else if (source.Count > 0)

{

channel3 = source.Dequeue();

progressBar3.Maximum = channel3.Dtki \* 1000;

requestProcessing3.Start();

channel3Pic.Fill = new SolidColorBrush(Colors.Yellow);

requestsRealCount++;

}

}

// Если все каналы заняты, а заявки в источчнике еще имеются, то

// их перенаправляем в накопитель...

while (source.Count > 0)

{

// ... только, если есть место в этом накопителе,

if (storageDevice.Count <= STORAGE\_DEVICE\_LIMIT - source.Count)

{

storageDevice.Enqueue(source.Dequeue());

if (storageDevice.Count == STORAGE\_DEVICE\_LIMIT) storageDevicePic.Fill = new SolidColorBrush(Colors.Orange);

}

// ... а если нет, то оставляем заявку необслуженной

else

{

failedRequests.Enqueue(source.Dequeue());

}

requestsRealCount++;

}

sourceLabel.Content = requestsRealCount.ToString();

storageLabel.Content = storageDevice.Count.ToString();

completedLabel.Content = completedRequests.Count.ToString();

failedLabel.Content = failedRequests.Count.ToString();

ch1Label.Content = requestsByChannel1.ToString();

ch2Label.Content = requestsByChannel2.ToString();

ch3Label.Content = requestsByChannel3.ToString();

requestsLeftLabel.Content = source.Count.ToString();

}

}

}

***Результат***

