**ОТЧЕТ**

по лабораторной работе №5

Исследования на имитационных моделях систем массового обслуживания

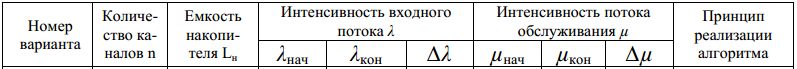
Выполнил студент группы АСОИ-122

Силков Е.В.

Проверил: Кушнер А.В.

***Цель работы:*** Проведение многофакторного расчетного эксперимента СМО на имитационной модели и обработка его результатов

***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;

using ZedGraph;

namespace IMS\_01

{

/// <summary>

/// Логика взаимодействия для Lab\_5.xaml

/// </summary>

public partial class Lab\_5 : Page

{

public const int STORAGE\_DEVICE\_LIMIT = 16;

public const double LAMBDA\_FIRST = 0.4;

public const double LAMBDA\_STEP = 0.22;

public const double LAMBDA\_LAST = 14.2;

public const double MU\_FIRST = 0.4 / 3;

public const double MU\_STEP = 0.18;

public const double MU\_LAST = MU\_FIRST + 12 \* MU\_STEP;

DispatcherTimer requestForming,

requestProcessing1,

requestProcessing2,

requestProcessing3,

timeRunningOut;

int requestsCount, requestsRealCount,

requestsByChannel1, requestsByChannel2, requestsByChannel3;

public double lambda = 0.4;

public double mu = 0.4 / 3;

List<double> lambdaList, muList, pDenied, pCompleted;

Queue<Request> source, storageDevice, completedRequests, failedRequests;

Request channel1, channel2, channel3;

Request someRequest;

public class Request // Запрос

{

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

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

public Request(double lambda, double mu)

{

Random r = new Random();

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

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

}

}

public Lab\_5()

{

InitializeComponent();

}

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

{

lambda = LAMBDA\_FIRST;

mu = 0.50;

requestsCount = requestsRealCount =

requestsByChannel1 = requestsByChannel2 = requestsByChannel3 = 0;

lambdaList = new List<double>();

muList = new List<double>();

pDenied = new List<double>();

pCompleted = new List<double>();

sourceLabel.Content = 0;

storageLabel.Content = 0;

completedLabel.Content = 0;

failedLabel.Content = 0;

ch1Label.Content = 0;

ch2Label.Content = 0;

ch3Label.Content = 0;

requestsLeftLabel.Content = 0;

lambdaLabel.Content = "λ= " + string.Format("{0:0.00}", lambda);

muLabel.Content = "μ= " + string.Format("{0:0.00}", mu);

requestForming = new DispatcherTimer();

requestProcessing1 = new DispatcherTimer();

requestProcessing2 = new DispatcherTimer();

requestProcessing3 = new DispatcherTimer();

timeRunningOut = new DispatcherTimer();

requestForming.Tick += RequestForming\_Tick;

requestProcessing1.Tick += RequestProcessing1\_Tick;

requestProcessing2.Tick += RequestProcessing2\_Tick;

requestProcessing3.Tick += RequestProcessing3\_Tick;

timeRunningOut.Tick += TimeRunningOut\_Tick;

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

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

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

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

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

requestsCount = requestsRealCount =

requestsByChannel1 = requestsByChannel2 = requestsByChannel3 = 0;

source = new Queue<Request>();

storageDevice = new Queue<Request>();

completedRequests = new Queue<Request>();

failedRequests = new Queue<Request>();

try

{

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

requestsCount = Convert.ToInt32(countTextBox.Text);

}

catch

{

int x;

if (!int.TryParse(timeTextBox.Text, out x)) x = 2500;

progressBarT.Maximum = x;

timeTextBox.Text = progressBarT.Maximum.ToString();

if (!int.TryParse(countTextBox.Text, out requestsCount)) requestsCount = 30;

countTextBox.Text = requestsCount.ToString();

}

someRequest = new Request(lambda, mu);

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))

{

if (lambda == 2.95)

{

if (mu >= MU\_LAST)

{

muList.Add(mu);

pCompleted.Add((double)completedRequests.Count / (failedRequests.Count + completedRequests.Count));

timeRunningOut.Stop();

MessageBox.Show("Готово!");

ZedGraphControl zedGraphControl1 = new ZedGraphControl();

wfh1.Child = zedGraphControl1;

zedGraphControl1.Width = zedGraphControl1.Height = 400;

zedGraphControl1.GraphPane.CurveList.Clear();

zedGraphControl1.GraphPane.AddCurve("",

lambdaList.ToArray(), pDenied.ToArray(), System.Drawing.Color.Red, SymbolType.None);

zedGraphControl1.AxisChange();

zedGraphControl1.Invalidate();

ZedGraphControl zedGraphControl2 = new ZedGraphControl();

wfh2.Child = zedGraphControl2;

zedGraphControl2.Width = zedGraphControl2.Height = 400;

zedGraphControl2.GraphPane.CurveList.Clear();

zedGraphControl2.GraphPane.AddCurve("",

muList.ToArray(), pCompleted.ToArray(), System.Drawing.Color.Red, SymbolType.None);

zedGraphControl2.AxisChange();

zedGraphControl2.Invalidate();

}

else

{

muList.Add(mu);

pCompleted.Add((double)completedRequests.Count / (failedRequests.Count + completedRequests.Count));

mu += MU\_STEP;

Initialize();

}

}

else

{

lambdaList.Add(lambda);

pDenied.Add((double)failedRequests.Count / (failedRequests.Count + completedRequests.Count));

if (lambda >= LAMBDA\_LAST)

{

lambda = 2.95;

mu = MU\_FIRST;

}

else lambda += LAMBDA\_STEP;

Initialize();

}

}

}

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(lambda, mu);

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();

}

public void Initialize()

{

requestsCount = requestsRealCount =

requestsByChannel1 = requestsByChannel2 = requestsByChannel3 = 0;

sourceLabel.Content = 0;

storageLabel.Content = 0;

completedLabel.Content = 0;

failedLabel.Content = 0;

ch1Label.Content = 0;

ch2Label.Content = 0;

ch3Label.Content = 0;

requestsLeftLabel.Content = 0;

source = new Queue<Request>();

storageDevice = new Queue<Request>();

completedRequests = new Queue<Request>();

failedRequests = new Queue<Request>();

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

requestsCount = Convert.ToInt32(countTextBox.Text);

someRequest = new Request(lambda, mu);

progressBarS.Maximum = someRequest.Dti \* 1000;

Update();

timeRunningOut.Start();

requestForming.Start();

lambdaLabel.Content = "λ= " + string.Format("{0:0.00}", lambda);

muLabel.Content = "μ= " + string.Format("{0:0.00}", mu);

}

}

}

***Результат***

