Національний технічний університет України

«Київський політехнічний інститут»

Кафедра обчислювальної техніки

Лабораторна робота №3

на тему:

«Системи обслуговування заявок»

Виконав

студент III курсу

групи ІО-83

Сірий Д.С.

Київ - 2011 р.

1. Алгоритм CBQ ( *Class-Based Queuing* ).

Класові дисципліни широко використовуються у випадках, коли різні види трафіку необхідно обробляти по-різному. Прикладом класової дисципліни може служити CBQ.

Коли трафік передається на обробку класовій дисципліні, він повинен бути віднесений до одного з класів (класифікований). Визначення приналежності пакету до того або іншого класу виконується фільтрами.

Фільтри, приєднані до дисципліни, повертають результат класифікації (клас пакету), після чого пакет передається в чергу, відповідну заданому класу. Кожний з класів у свою чергу може складатися з підкласів і мати свій набір фільтрів для виконання точнішої класифікації своєї частини трафіку. Інакше пакет обслуговується дисципліною черги класу.

Крім того, в більшості випадків класові дисципліни виконують шейпінг (формування) трафіку, з метою пере впорядковування пакетів і управління швидкістю їх передачі. Це абсолютно необхідно у разі пере направлення трафіку з високошвидкісного інтерфейсу (наприклад, Еthernet) на повільний (наприклад, модем).

Це дозволяє різним прикладним програмам спільно використовувати одну і ту ж мережу, причому кожна з них пред'являє свої специфічні мінімальні вимоги до ширини полоси пропускання або до затримки.

# Лістинг програми

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace os\_lab\_3\_cs

{

class GenFactory

{

public static IEnumerator<Package> getGenerator(int minSize, int maxSize,

int minPriority, int maxPriority)

{

Random rnd = new Random();

while (true)

{

yield return new Package(rnd.Next(minSize, maxSize),

rnd.Next(minPriority, maxPriority));

}

}

}

class Package

{

public Package(int size, int priority) {

Size = size;

Priority = priority;

Time = 0;

}

public int Size { get; set; }

public int Time { get; set; }

public int Priority { get; set; }

}

abstract class Abstract\_CBQ

{

abstract public void push(Package pac);

abstract public void tick();

public Predicate<Package> Pred { get; protected set; }

public virtual int Time { get; protected set; }

public virtual int Counter { get; protected set; }

public virtual int DownTime { get; protected set; }

}

class CBQ : Abstract\_CBQ

{

private int buffSize;

private int intensity;

private int buffFree;

private Queue<Package> packages;

public CBQ(Predicate<Package> pred, int intensity, int buffSize)

{

this.buffSize = buffSize;

this.intensity = intensity;

Pred = pred;

buffFree = buffSize;

Counter = 0;

Time = 0;

DownTime = 0;

packages = new Queue<Package>();

}

public override void push(Package pac)

{

pac.Time = 0;

if (buffFree >= pac.Size)

{

buffFree -= pac.Size;

packages.Enqueue(pac);

}

}

public override void tick()

{

int proc = intensity;

foreach (var pac in packages)

{

pac.Time++;

}

buffFree += intensity;

if (packages.Count == 1)

{

int x = 1;

int y = 2 + x;

}

if (buffFree > buffSize)

{

buffFree = buffSize;

}

while (proc > 0)

{

if (packages.Count == 0)

{

DownTime++;

return;

}

Package pac = packages.Peek();

pac.Size = pac.Size - proc;

if (pac.Size <= 0)

{

packages.Dequeue();

Time += pac.Time;

Counter++;

}

proc = -pac.Size;

}

}

}

class CBQs : Abstract\_CBQ

{

List<Abstract\_CBQ> cbqs;

public CBQs(Predicate<Package> pred, List<Abstract\_CBQ> cbqs)

{

Pred = pred;

this.cbqs = cbqs;

}

public override void push(Package pac)

{

var cbq = cbqs.Find(q => q.Pred(pac));

if (cbq != null)

{

cbq.push(pac);

}

}

public override void tick()

{

foreach (var q in cbqs)

{

q.tick();

}

}

public override int Time

{

get

{

return cbqs.Select(q => q.Time).Sum();

}

}

public override int Counter

{

get

{

return cbqs.Select(q => q.Counter).Sum();

}

}

public override int DownTime

{

get

{

return cbqs.Select(q => q.DownTime).Sum();

}

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace os\_lab\_3\_cs

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

chart1.Series.Add("");

for (int intensity = 50; intensity > 0 ; intensity--)

{

var qs1 = new List<Abstract\_CBQ>();

qs1.Add(new CBQ(pac => pac.Size < 10, 2, 150));

qs1.Add(new CBQ(pac => pac.Size >= 10, 6, 160));

var qs2 = new List<Abstract\_CBQ>();

qs2.Add(new CBQ(pac => pac.Priority <= 3, 7, 170));

qs2.Add(new CBQs(pac => pac.Priority > 3, qs1));

var cbqs = new CBQs(pac => true, qs2);

int delay = intensity;

var gen = GenFactory.getGenerator(1, 50, 0, 10);

var rnd = new Random();

for (int i = 0; i < 10000; ++i)

{

if (--delay == 0)

{

gen.MoveNext();

cbqs.push(gen.Current);

delay = intensity;

}

cbqs.tick();

}

int tavrg = cbqs.Time / cbqs.Counter;

chart1.Series[0].Name = "Среднее время ожидания от интенсивности";

chart1.Series[0].ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;

chart1.Series[0].Points.AddXY((50/intensity), tavrg);

chart1.Series[1].Name = "Среднее время простоя от интенсивности";

chart1.Series[1].ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;

chart1.Series[1].Points.AddXY((50 / intensity), cbqs.DownTime/300.0);

Console.WriteLine("{0} : {1}", cbqs.Time, cbqs.Counter);

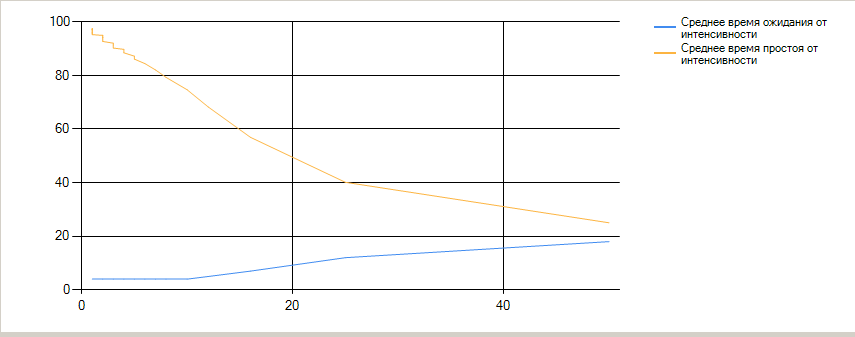
}

}

}

}

# Графіки



# Висновки

З графіків видно, що при збільшенні інтенсивності вхідного потоку процент простою черги падає, а середній час очікування в черзі збільшується.