Міністерство освіти і науки України

Національний технічний університет України

«Київський Політехнічний Інститут»

Факультет прикладної математики

Кафедра Спеціалізованих комп’ютерних систем

Лабораторна робота №3

З дисципліни «Системне програмне забезпечення» :

«Дослідження дисциплін обслуговування заявок при обмежених ресурсах»

Варіант №16

Виконав:

студент групи КВ-92

Степанюк М.Ф.

Перевірив:

Симоненко В. П.

Київ 2012

***Алгоритм RED***

Одним з можливих підходів при вирішенні проблеми перевантаження є алгоритм RED (Random Early Detection). RED дозволяє маршрутизатору відкидати пакети, навіть коли в черзі ще є місце. Головною метою алгоритму RED є виключення ситуації, коли декілька ТСР - потоків перевантажуються майже одночасно і потім синхронно починають процедуру відновлення..

***Лістинг програми:***

using System;

namespace SPZ3

{

public class thread

{

public int PerfQaunt;

public thread Next;

public DateTime dt;

public TimeSpan ts;

public int q\_len;

public thread()

{

PerfQaunt = 0;

Next = null;

dt = new DateTime();

ts = new TimeSpan();

q\_len = 0;

}

public thread(thread obj)

{

Next = null;

PerfQaunt = obj.PerfQaunt;

dt = obj.dt;

ts = obj.ts;

q\_len = obj.q\_len;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Timers;

using System.IO;

namespace SPZ3

{

class scheduler

{

public thread head;

public thread tail;

public int min\_lim;

public int max\_lim;

public int qu\_len;

private int cons\_qu;

public Timer Consumer;

public Timer Producer;

static List<thread> Current = new System.Collections.Generic.List<thread>();

static List<thread> Thrown = new System.Collections.Generic.List<thread>();

static List<thread> Performed = new System.Collections.Generic.List<thread>();

public scheduler()

{

head = null;

tail = null;

min\_lim = 8;

max\_lim = 20;

cons\_qu = 1;

qu\_len = 0;

}

public void push\_back(thread ptr)

{

thread buf = new thread(ptr);

buf.q\_len = qu\_len;

if (head == null)

{

head = buf;

tail = head;

cons\_qu = ptr.PerfQaunt;

return;

}

tail.Next = buf;

tail = buf;

tail.Next = null;

return;

}

public void pop\_front()

{

thread ptr = head;

if (qu\_len == 0)

return;

head = head.Next;

ptr.Next = null;

qu\_len--;

if (head == null)

{

tail = null;

return;

}

cons\_qu = head.PerfQaunt;

return;

}

public void GenerateQueue ()

{

for (int i = 0; i < Thrown.ToArray().Length; i++)

Current.Add(Thrown.ToArray()[i]);

Thrown.Clear();

Random rnd = new Random();

int cnt = rnd.Next(1, 4);

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

{

thread ptr = new thread();

ptr.PerfQaunt = rnd.Next(1,4);

ptr.dt = DateTime.Now;

Current.Add(ptr);

}

}

public void ThrowPackage(int mode)

{

List<thread> buf = new System.Collections.Generic.List<thread>();

for (int i = 1; i < Current.ToArray().Length; i++)

buf.Add(Current.ToArray()[i]);

thread ptr = Current.First();

if (mode == 1)

Thrown.Add(ptr);

Current.Clear();

for (int i = 0; i < buf.ToArray().Length; i++)

Current.Add(buf.ToArray()[i]);

}

public void SchStart()

{

try

{

Producer = new Timer();

Producer.Elapsed += new ElapsedEventHandler(TimerProducer);

Producer.Enabled = true;

Producer.Interval = 1000;

Producer.Start();

Consumer = new Timer();

Consumer.Elapsed += new ElapsedEventHandler(TimerConsumer);

Consumer.Interval = 1000;

Consumer.Start();

while (Performed.ToArray().Length < 20 )

{

//do nothing

}

Producer.Stop();

Consumer.Stop();

Console.Write("\n");

StreamWriter fl\_w = new StreamWriter(new FileStream("spz3.txt", FileMode.Create));

for (int i = 0; i < Performed.ToArray().Length; i++)

{

string str = Performed.ToArray()[i].q\_len.ToString() + " " + Performed.ToArray()[i].ts.TotalMilliseconds.ToString();

fl\_w.WriteLine(str);

Console.WriteLine("{0} packages length {1}", Performed.ToArray()[i].ts.TotalMilliseconds, Performed.ToArray()[i].q\_len);

}

fl\_w.Close();

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

}

public void PropAdd()

{

Random rnd = new Random();

double val;

if (qu\_len <= min\_lim)

val = rnd.Next(0, min\_lim);

else

val = rnd.Next(qu\_len, max\_lim);

val = val - min\_lim;

if (val < 0)

val = 0;

double prop = val / (max\_lim - min\_lim);

Console.Write("propability of adding process is {0:F4} - ", prop);

if (prop > 0.6)

{

Console.Write("packet wasn't added. Current queue length {0}\n", qu\_len);

ThrowPackage(1);

return;

}

thread ptr = new thread (Current.First());

push\_back(ptr);

qu\_len++;

ThrowPackage(0);

Console.WriteLine(" added new process {0} with perf\_quant {1} ", qu\_len,ptr.PerfQaunt);

}

public void TimerProducer(object source, ElapsedEventArgs e)

{

Producer.Stop();

GenerateQueue();

for (int i = 0; i < Current.ToArray().Length; i++)

PropAdd();

Producer.Start();

return;

}

public void TimerConsumer(object source, ElapsedEventArgs e)

{

Consumer.Stop();

if (cons\_qu > 0)

cons\_qu--;

else

{

thread ptr = new thread(head);

ptr.ts = DateTime.Now - ptr.dt;

Performed.Add(ptr);

pop\_front();

Console.WriteLine("router took process to perform, current queue length {0}", qu\_len);

}

Consumer.Start();

}

static void Main(string[] args)

{

scheduler sch = new scheduler();

sch.SchStart();

}

}

}

