Міністерство освіти і науки України

Національний технічний університет України

„Київський політехнічний інститут”

Факультет прикладної математики

Кафедра спеціалізованих комп’ютерних систем

***Лабораторна робота №1***

*З дисципліни «Операційні системи»*

*На тему: «Планування процесів»*

**Виконав:**

студент ІІІ курсу

групи КВ-92

Степанюк М. Ф.

**Перевірив:**

Зайцев В. Г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ 2012

**Варіант №9: Черги із зворотним зв'язком**

Черг такого типу може бути більше ніж дві, і процеси можуть переміщуватися між ними. Перехід між чергами називають із зворотнім зв'язком. Визначення стратегії планування з багатьма чергами і зворотним зв'язком повинне включати:

1. Кількість черг.

2. Стратегія планування для кожної черги.

3. Спосіб визначення процесу, що переміщується в іншу чергу.

4. Спосіб визначення черги, в яку повинен бути поміщений новий процес (**в завданні будемо вважати, що черга вже визначена!**).

Уявимо собі, наприклад, стратегію з трьома чергами Q0, Q1 і Q2. Умови наступні:

1. Черга Q0 має вищий пріоритет у порівнянні з іншими (тобто, якщо процес знаходиться в Q0, тоді процеси з решти черг не беруться до уваги).

2. Черга Q1 має вищий пріоритет, ніж Q2.

3. Черга Q0 обслуговується згідно [RR](file:///E:\Project\ОС\4\lecture.html#rr#rr) алгоритму з величиною кванта в 50 мс.

4. Черга Q1 обслуговується згідно [RR](file:///E:\Project\ОС\4\lecture.html#rr#rr) алгоритму з величиною кванта в 100 мс.

5. Черга Q2 обслуговується згідно FCFS алгоритму.

6. Новий процес надходить в Q0.

7. Якщо процес з черги Q0 не завершується при надходженні кванта часу (50 мілісекунд), то він витісняється і передається в чергу Q1.

8. Якщо процес з черги Q1 не завершується при надходженні кванта часу (100 мілісекунд), то він витісняється і передається в чергу Q2.

9. Не можна повернутися в чергу з вищим пріоритетом.

**Для моделювання обрати 3** черги, одна з яких обслуговується за алгоритмім FCFS, а дві інші – за алгоритмім Round-Robin.

**FCFS (**First Come First Serve) – першим прийшов – першим обслуговується.

Це – невитісняльна стратегія. Процеси виконуються в послідовності їх надходження до системи. i-й процес може бути запущений на виконання тільки після завершення (i-1)-го процесу. Якщо в момент надходженняi-го процесу (i-1)-й процес ще не закінчився, тоді i-й процес переходить в стан очікування до момента вивільнення процесора.

## Алгоритм планування Round-Robin

RR (Round-Robin) – кругове (карусельне) планування– стратегія, коли кожний процес отримує у використання квант процесорного часу. Зазвичай, він складає близько 100 мілісекунд. Таким чином, процеси не повинні будуть чекати довше, ніж довжина кванту, що помножена на кількість процесів, які виконуються в системі.

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;

using System.Diagnostics;

using System.Threading;

namespace OS\_lab1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

info.Clear();

logger.Clear();

List<Descriptor> initial = new List<Descriptor>();

Queue<int> Q0 = new Queue<int>();

Queue<int> Q1 = new Queue<int>();

Queue<int> Q2 = new Queue<int>();

int tiks\_counter = 0;

int process\_count = (int)numericUpDown1.Value;

// filling up initial

int mod = 0;

Descriptor ts\_tmp;

for (int i = 0; i < process\_count; i++)

{

ts\_tmp = new Descriptor(i, mod);

initial.Add(ts\_tmp);

mod = ts\_tmp.begin;

}

int new\_proc = 0;

int kvant0 = 50;

int kvant1 = 100;

int tmp;

string output = "";

while (tiks\_counter <= 100000)

{

// --------- scheduler emulation here -----------------------

// 1) Adding new process to Q0 if it is time for it

if ((new\_proc < process\_count) && (initial[new\_proc].begin == tiks\_counter))

{

output += "P" + initial[new\_proc].id.ToString() + " START at " + tiks\_counter.ToString() + "\n";

Q0.Enqueue(initial[new\_proc].id);

new\_proc++;

}

//Q0 handling with quantum = 50

if (Q0.Count != 0)

if (kvant0 != 0)

{

//there is kvant time. so first process gains -1 work todo, all other gain +1 delay

initial[Q0.Peek()].todo--;

output += "Q0: P" + initial[Q0.Peek()].id.ToString() + " -1 TODO:" + initial[Q0.Peek()].todo.ToString() + " k0:" + kvant0.ToString() + " d:" + initial[Q0.Peek()].delay.ToString() + "\n";

foreach (int i in Q0) initial[i].delay++;

initial[Q0.Peek()].delay--;

foreach (int i in Q1) initial[i].delay++;

foreach (int i in Q2) initial[i].delay++;

kvant0--;

//but if first process has already done his work we delete it from q0 and set kvant0 to 50.

if (initial[Q0.Peek()].todo == 0)

{

output += "Q0: P" + initial[Q0.Peek()].id.ToString() + " FINISH at " + tiks\_counter.ToString() + "\n";

initial[Q0.Peek()].end = tiks\_counter;

Q0.Dequeue();

kvant0 = 50;

}

}

else

{

//kvant = 0. so the first process in Q0 hadn't done his work in kvant, moving it to Q1

output += "P" + initial[Q0.Peek()].id.ToString() + " Q0 --> Q1\n";

Q1.Enqueue(Q0.Dequeue());

tiks\_counter--; //considering that this operation doesn't take any time.

kvant0 = 50;

}

else // if q0 is empty we go here

{

if (Q1.Count != 0)

if (kvant1 != 0)

{

//there is kvant time. so first process gains -1 work todo, all other gain +1 delay

initial[Q1.Peek()].todo--;

output += "Q1: P" + initial[Q1.Peek()].id.ToString() + " -1 TODO:" + initial[Q1.Peek()].todo.ToString() + " k1:" + kvant1.ToString() + " d:" + initial[Q1.Peek()].delay.ToString() + "\n";

foreach (int i in Q1) initial[i].delay++;

initial[Q1.Peek()].delay--;

foreach (int i in Q2) initial[i].delay++;

kvant1--;

//but if first process has already done his work we delete it from q1 and set kvant1 to 100.

if (initial[Q1.Peek()].todo == 0)

{

output += "Q1: P" + initial[Q1.Peek()].id.ToString() + " FINISH at " + tiks\_counter.ToString() + "\n";

initial[Q1.Peek()].end = tiks\_counter;

Q1.Dequeue();

kvant1 = 100;

}

}

else

{

//kvant = 0. so the first process in Q1 hadn't done his work in kvant, moving it to Q2

output += "P" + initial[Q1.Peek()].id.ToString() + " Q1 --> Q2\n";

Q2.Enqueue(Q1.Dequeue());

tiks\_counter--; //considering that this operation doesn't take any time.

kvant1 = 100;

}

else // and now, all queues 0 and 1 are empty, we can execute the 2 queue

{

if (Q2.Count != 0)

{

initial[Q2.Peek()].todo--;

output += "Q2: P" + initial[Q2.Peek()].id.ToString() + " -1 TODO:" + initial[Q2.Peek()].todo.ToString() + " d:" + initial[Q2.Peek()].delay.ToString() + "\n";

//foreach (int i in Q1) initial[Q1.Peek()].delay++;

//foreach (int i in Q0) initial[Q0.Peek()].delay++;

foreach (int i in Q2) initial[i].delay++;

initial[Q2.Peek()].delay--;

//but if first process has already done his work we delete it from q1 and set kvant1 to 100.

if (initial[Q2.Peek()].todo == 0)

{

output += "Q2: P" + initial[Q2.Peek()].id.ToString() + " FINISH at " + tiks\_counter.ToString() + "\n";

initial[Q2.Peek()].end = tiks\_counter;

Q2.Dequeue();

}

}

}

}

tiks\_counter++;

}

//debug output + correct formatting

double sum = 0;

foreach (Descriptor t in initial)

{

info.Text += string.Format(" {0,2} {1,5} {2,5} {3,5} {4,5}\n", t.id, t.begin.ToString(), t.length.ToString(), t.end.ToString(), t.delay.ToString());

sum += t.delay;

}

sum = sum / initial.Count;

logger.Text = output;

info.Text += "Середній час очікування: " + sum.ToString();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

}

//============================================ R A N D O M

static class rnd

{

public static Random r = new Random();

}

//============================================ T H R E A D S T A T S

class Descriptor

{

public Descriptor(int i, int modifier)

{

id = i;

begin = rnd.r.Next(150) + modifier+1;

length = rnd.r.Next(200)+ 1;

todo = length;

}

public int id;

public int begin;

public int length;

public int todo;

public int end = 0;

public int delay = 0;

}

}