Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота №3

з дисципліни «Операційні системи»

Виконала:

студентка гр. ІС-71

Новаківська Катерина

Перевірив:

Дифучин А. Ю.

Київ – 2020

1. Опис дисципліни обслуговування:

Базовою ідеєю алгоритму є забезпечення рівнозначного доступу кожної з черг до ресурсів процесора - кожен раз, коли процесор звільняється, планувальник циклічно вибирає чергу, з якої приймається пакет на обслуговування.

Якщо в черзі, коли до неї звертається планувальник, відсутній пакет, то планувальник звертається до наступної по порядку черги.

1. Код програми:

public partial class Charts : Form

{

public Charts()

{

InitializeComponent();

FillChart();

}

private void chart1\_Click(object sender, EventArgs e)

{

//FillChart();

}

private void chart2\_Click(object sender, EventArgs e)

{

//FillChart();

}

public void FillChart()

{

var roundRobin = new RoundRobin();

Random rand = new Random();

var list = new List<Result>();

var list1 = new List<Result>();

int quantum = 3;

for (int i = 0; i < 32; i++)

{

var processCount = rand.Next(1,10);

int[] processes = new int[processCount];

for (int j = 0; j < processes.Length; j++)

{

processes[j] = j + 1;

}

int[] burstTime = new int[processCount];

for (int j = 0; j < burstTime.Length; j++)

{

burstTime[j] = rand.Next(1,10);

}

var results = roundRobin.FindAverageTime(processes, processCount, burstTime, quantum);

var element = new Result(results[0], results[1], processCount);

list.Add(element);

}

var query = list.Distinct(new IntensivityComparer()).OrderBy(x => x.intensivity);

chart1.Titles.Add("Average waiting to intensivity");

chart1.Series["Waiting"].ChartType = SeriesChartType.Spline;

chart2.Titles.Add("Intensivity to procent of lay-up");

chart2.Series["Intensivity"].ChartType = SeriesChartType.Spline;

chart3.Titles.Add("Tasks to average time in turn");

chart3.Series["Procent"].ChartType = SeriesChartType.Spline;

var query1 = list.Distinct(new TimeComparer()).OrderBy(x => x.averageWaitingTime);

foreach (var el in query)

{

chart1.Series["Waiting"].Points.AddXY(100/el.intensivity, Math.Round(el.averageWaitingTime, 2));

chart2.Series["Intensivity"].Points.AddXY(100/el.intensivity, (el.intensivity-quantum)\*100/quantum);

chart3.Series["Procent"].Points.AddXY((el.intensivity - quantum), Math.Round(el.averageWaitingTime, 2));

}

}

}

public class Result

{

public float averageWaitingTime { get; set; }

public float averageTurnAroundTime { get; set; }

public int intensivity { get; set; }

public Result(float averageWaitingTime, float averageTurnAroundTime, int intensivity)

{

this.averageWaitingTime = averageWaitingTime;

this.averageTurnAroundTime = averageTurnAroundTime;

this.intensivity = intensivity;

}

}

internal class IntensivityComparer : IEqualityComparer<Result>

{

public bool Equals(Result x, Result y)

{

if (string.Equals((x.intensivity).ToString(), (y.intensivity).ToString(), StringComparison.OrdinalIgnoreCase))

{

return true;

}

return false;

}

public int GetHashCode(Result obj)

{

return obj.intensivity.GetHashCode();

}

}

internal class TimeComparer : IEqualityComparer<Result>

{

public bool Equals(Result x, Result y)

{

if (string.Equals((x.averageWaitingTime).ToString(), (y.averageWaitingTime).ToString(), StringComparison.OrdinalIgnoreCase))

{

return true;

}

return false;

}

public int GetHashCode(Result obj)

{

return obj.averageWaitingTime.GetHashCode();

}

}

public class RoundRobin

{

// знаходження часу очікування

static void FindWaitingTime(int[] processes, int n, int[] bt, int[] wt, int quantum)

{

// час виконання завдання

int[] rem\_bt = new int[n];

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

rem\_bt[i] = bt[i];

int t = 0; // поточний час

// поки всі завдання не виконані

while (true)

{

bool done = true;

// прохід всіх процесів в потоці

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

{

if (rem\_bt[i] > 0)

{

//процес в очікуванні

done = false;

if (rem\_bt[i] > quantum)

{

// збільшення часу

t += quantum;

// зменшення критичного часу

rem\_bt[i] -= quantum;

}

else

{

// час, за який виконується процес

t = t + rem\_bt[i];

// час очікування є меншим на час виконання

wt[i] = t - bt[i];

// час, що лишився

rem\_bt[i] = 0;

}

}

}

// всі процеси виконані

if (done == true)

break;

}

}

// обрахування часу в черзі

static void FindTurnAroundTime(int[] processes, int n, int[] bt, int[] wt, int[] tat)

{

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

tat[i] = bt[i] + wt[i];

}

// обрахування середнього часу

public float[] FindAverageTime(int[] processes, int n, int[] bt, int quantum)

{

int[] wt = new int[n];

int[] tat = new int[n];

float[] results = new float[2];

int total\_wt = 0, total\_tat = 0;

// знаходження асу очікування

FindWaitingTime(processes, n, bt, wt, quantum);

// обрахування часу в черзі

FindTurnAroundTime(processes, n, bt, wt, tat);

// повний час

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

}

results[0] = (float) total\_wt / (float) n;

results[1] = (float)total\_tat / (float)n;

return results;

}

}

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

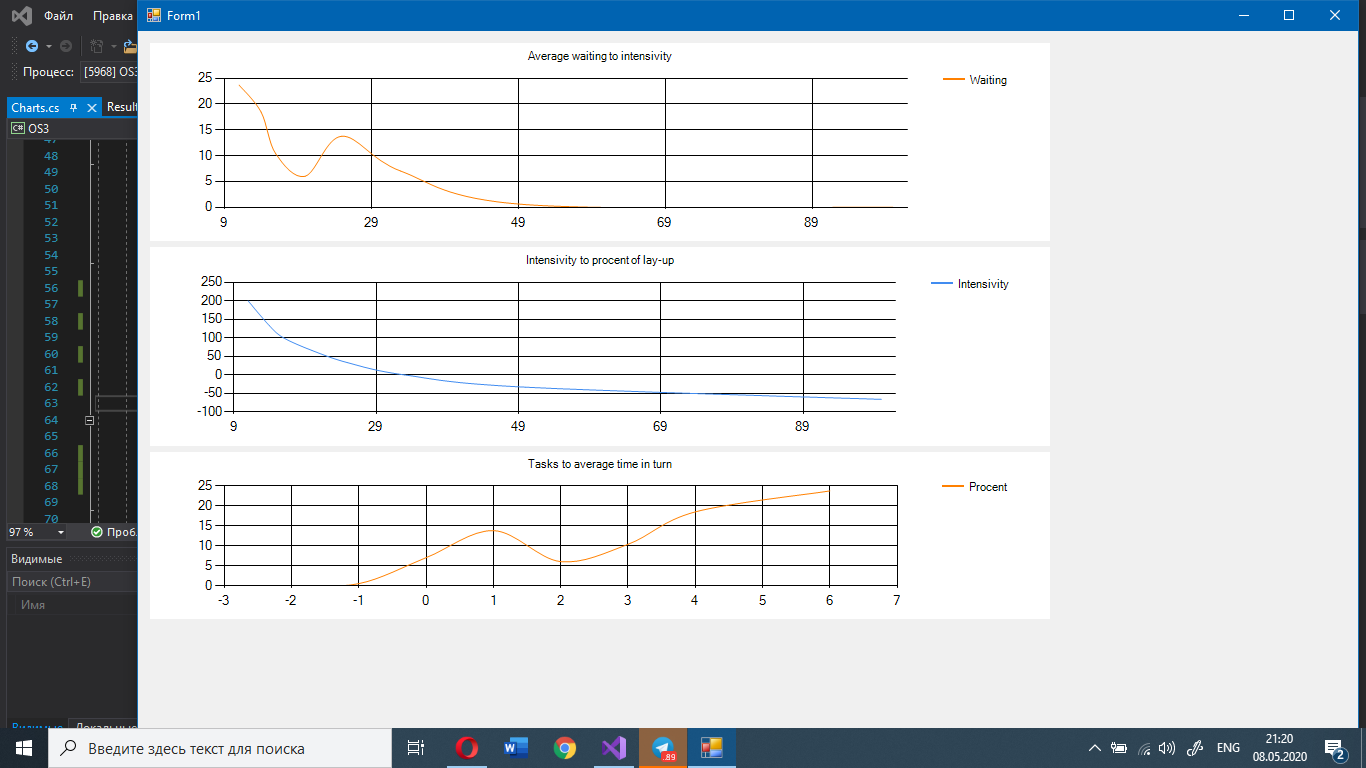
Application.Run(new Charts());

}

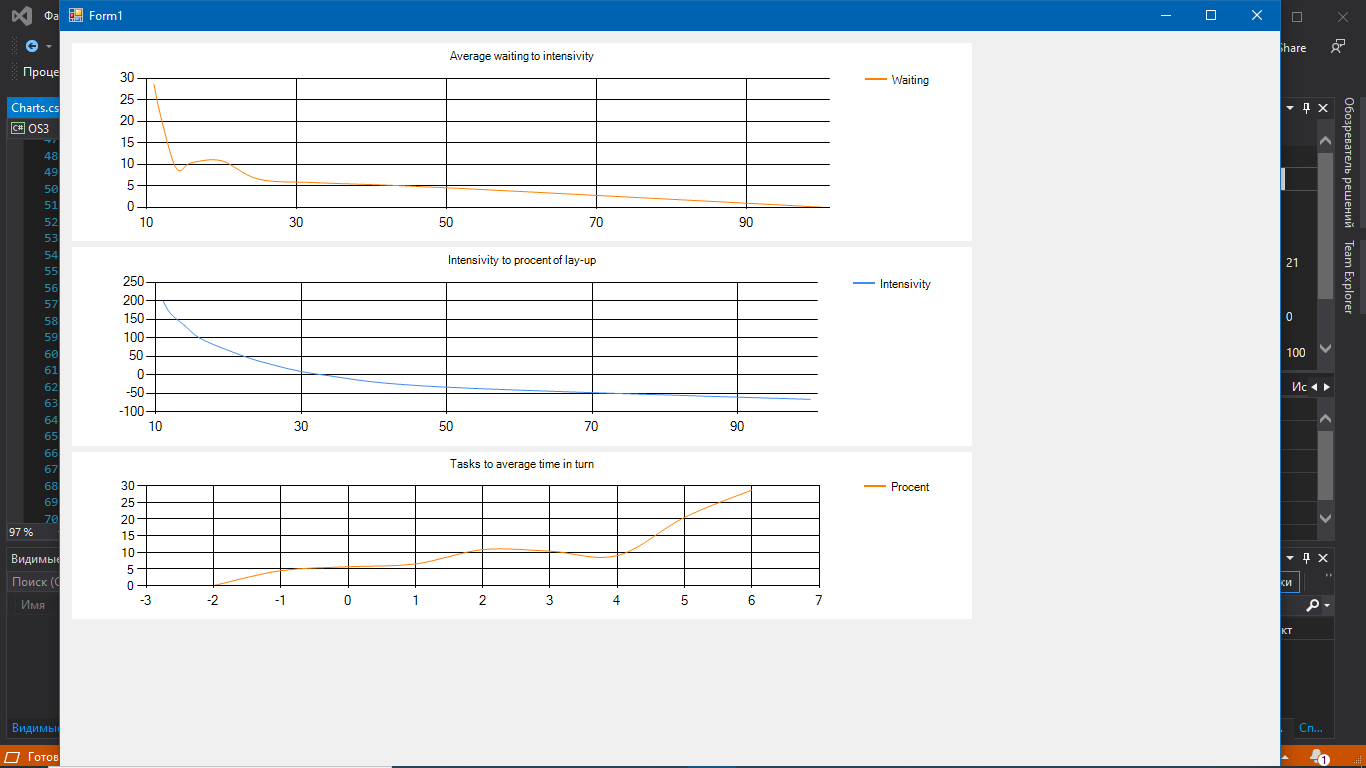
}

1. Графіки:

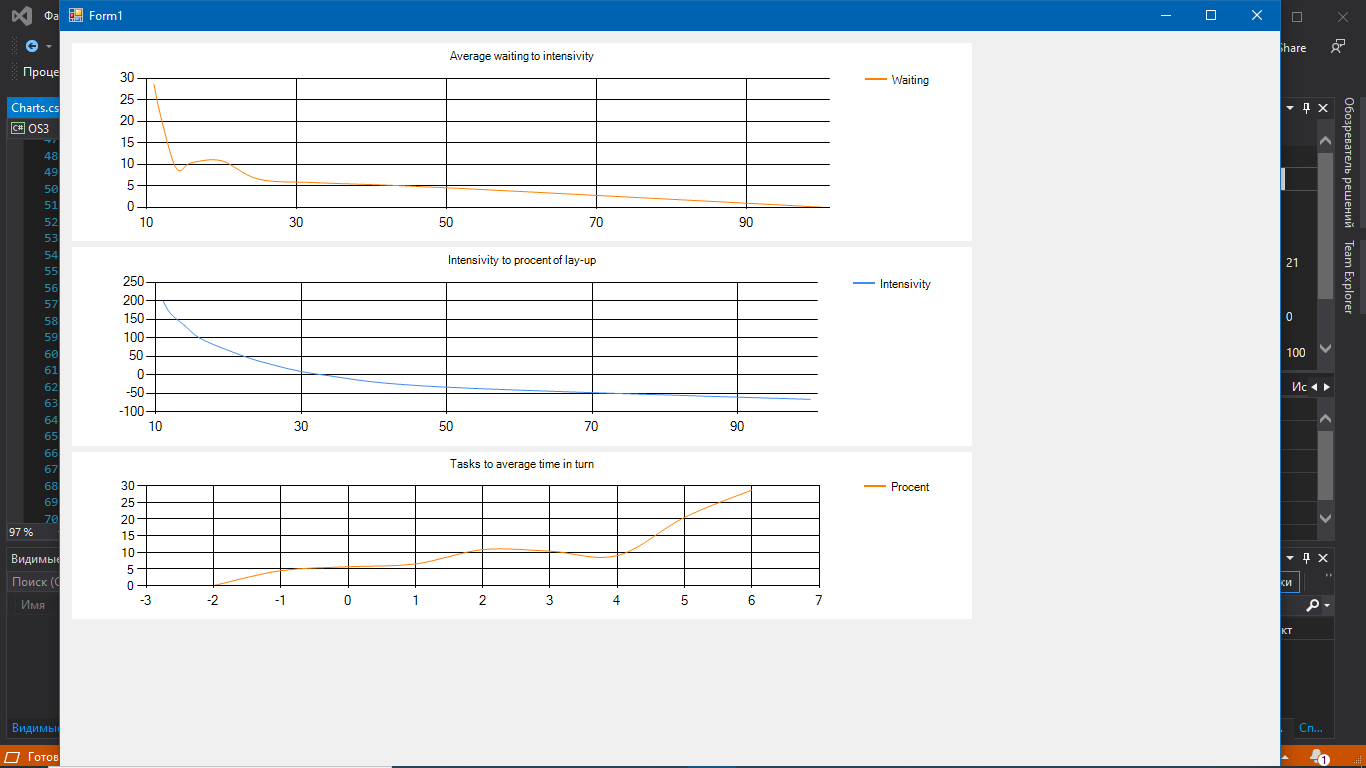
*Графік залежності середнього часу очікування від інтенсивності вхідного потоку заявок:*



*Графік залежності проценту простою ресурсу від інтенсивності вхідного потоку:*



*Графік залежності кількості заявок від часу очікування:*



1. Плюси та мінуси досліджуваної дисципліни:

Плюси:

* відносно невелика складність алгоритму
* дозволяє організувати зрозуміле, передбачуване і стабільне балансування

Мінуси:

* в реальному обладнанні - RR не забезпечує принцип справедливого розподілу ресурсів для випадків, коли пакети мають змінну довжину
* при занадто малій величині кванта часу і, відповідно, занадто частому перемиканні контексту, накладні витрати на перемикання різко знижують продуктивність системи
* при балансуванні алгоритм не враховує кількість активних сполук, і навіть якщо один з вузлів буде повністю завантажений, він все одно отримає нові запити.