ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ОЛЕСЯ ГОНЧАРА

Факультет прикладної математики

Кафедра математичного моделювання та технологій програмування

Лабораторна робота №5

З дисципліни «Комп’ютерне моделювання систем та процесів»

Виконавець

Студент групи ПА–19–2

Ільяшенко Єгор

Дніпро

2022

Мета: Реалізувати свій власний застосунок gpss із своїм варіантом.

Изображение выглядит как стол

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Для виконання цієї роботи, я обрав мову програмування C#. Платформа розробки: Console Application.

Порівняння виводу програми gpss та своєї програми

Симуляція, тривалістю 480 хвилин.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Програма виводить дані для станків та їх черг.

Для станків виводяться наступні дані:

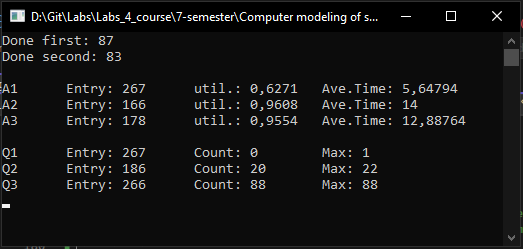
1. Кількість деталей, яка поступила на станок.
2. Утилізація (зайнятість) станку.
3. Середній час обробки деталі

Для черг маємо наступній вивід:

1. Кількість деталей, які поступили у чергу
2. Теперішня кількість деталей у черзі
3. Максимальна кількість деталей у черзі

Оскільки програма генерує час генерації, або обробки, результати кожен раз трохи відрізняються. Але із зображень видно, що на одних і тих же даних gpss та програма видають доволі схожі результати.

Також можна провести симуляцію для повної робочої неділі (2400 хвилин):



Висновок: Виконуючи лабораторну роботу №5, я розробив власну версію програми gpss на своєму варіанті. Програма дозволяє симулювати роботу цеху з трьох верстатів, які оброблюють 2 різних типи деталей.

**Код програми**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Runtime.InteropServices;

using System.Xml.Linq;

namespace Lab

{

internal class Program

{

#region Old

class GPSSEnvironment

{

public Dictionary<string, Queue<Element>> queue;

public List<ICommand> commands = new List<ICommand>();

public List<Element> active\_elements = new List<Element>();

int model\_time = -1;

Action<string[]> return\_callback = null;

public GPSSEnvironment(string[] commands\_list, Action<string[]> return\_callback)

{

queue = new Dictionary<string, Queue<Element>>();

queue.Add("GenerationQueue", new Queue<Element>());

this.return\_callback = return\_callback;

foreach(string command in commands\_list)

{

if(string.IsNullOrEmpty(command) || command[0] == ';')

continue;

if (command.Contains("GENERATE")) commands.Add(new Generate(this, command));

else if (command.Contains("QUEUE")) commands.Add(new Queue(this, command));

else if (command.Contains("SEIZE")) commands.Add(new Seize(this, command));

else if (command.Contains("DEPART")) commands.Add(new Depart(this, command));

else if (command.Contains("ADVANCE")) commands.Add(new Advance(this, command));

else if (command.Contains("RELEASE")) commands.Add(new Release(this, command));

else if (command.Contains("TIME")) model\_time = int.Parse(command.Split('|')[1]);

}

if (model\_time == -1)

return\_callback(null);

for (int i = 0; i < model\_time; i++)

commands[i].Execute(i);

}

}

class Generate : ICommand

{

GPSSEnvironment environment = null;

int middle\_time = -1;

int variety\_time = -1;

bool has\_variety = false;

int execution\_time = -1;

void ResetExecutionTime(int current\_time)

{

execution\_time = current\_time + middle\_time;

Random rnd = new Random();

if (has\_variety)

execution\_time += rnd.Next(-variety\_time, variety\_time);

}

public void Execute(int time)

{

if (time != execution\_time)

return;

ResetExecutionTime(execution\_time);

}

public Generate(GPSSEnvironment env, string attributes)

{

List<string> attributes\_list = new List<string>(attributes.Split('|'));

attributes\_list.RemoveAt(0);

this.environment = env;

middle\_time = int.Parse(attributes\_list[1]);

if (attributes\_list.Count == 2)

{

variety\_time = int.Parse(attributes\_list[2]);

has\_variety = true;

}

ResetExecutionTime(0);

}

}

class Queue : ICommand

{

string queue\_name = null;

GPSSEnvironment environment = null;

public void Execute(int time)

{

}

public Queue(GPSSEnvironment env, string attributes)

{

this.environment = env;

string[] attributes\_ = attributes.Split('|');

queue\_name = attributes\_[1];

}

}

class Seize : ICommand

{

string worker\_name = null;

Element element = null;

GPSSEnvironment environment = null;

public void Execute(int time)

{

}

public Seize(GPSSEnvironment env, string attributes)

{

this.environment = env;

worker\_name = attributes.Split('|')[1];

}

}

class Depart : ICommand

{

GPSSEnvironment environment = null;

public void Execute(int time)

{

}

public Depart(GPSSEnvironment env, string attributes)

{

this.environment = env;

}

}

class Advance : ICommand

{

GPSSEnvironment environment = null;

public void Execute(int time)

{

}

public Advance(GPSSEnvironment env, string attributes)

{

this.environment = env;

}

}

class Release : ICommand

{

GPSSEnvironment environment = null;

public void Execute(int time)

{

}

public Release(GPSSEnvironment env, string attributes)

{

this.environment = env;

}

}

interface ICommand

{

void Execute(int time);

}

class Element

{

}

public static string InputFilePath = $"{Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location)}/Input.txt";

public static string OutputFilePath = $"{Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location)}/Output.txt";

#endregion

static void Main(string[] args)

{

//if (!File.Exists(InputFilePath))

// return;

//string[] commands = File.ReadAllLines(InputFilePath);

//GPSSEnvironment gpss = new GPSSEnvironment(commands, result => File.WriteAllLines(OutputFilePath, result));

GPSS gpss = new GPSS(2400);

Console.WriteLine($"Done first: {gpss.first\_done}");

Console.WriteLine($"Done second: {gpss.second\_done}");

Console.WriteLine();

Console.WriteLine($"A1\tEntry: {gpss.a1.entries}\tutil.: {Math.Round(1f \* gpss.a1.busy\_iterations / (gpss.a1.busy\_iterations + gpss.a1.free\_iterations),4)}\tAve.Time: {(1f \* gpss.a1.first.overall\_time + gpss.a1.second.overall\_time)/(gpss.a1.first.overall\_timestamps\_count+gpss.a1.second.overall\_timestamps\_count)}");

Console.WriteLine($"A2\tEntry: {gpss.a2.entries}\tutil.: {Math.Round(1f \* gpss.a2.busy\_iterations / (gpss.a2.busy\_iterations + gpss.a2.free\_iterations),4)}\tAve.Time: {(1f \* gpss.a2.first.overall\_time + gpss.a2.second.overall\_time)/(gpss.a2.first.overall\_timestamps\_count+gpss.a2.second.overall\_timestamps\_count)}");

Console.WriteLine($"A3\tEntry: {gpss.a3.entries}\tutil.: {Math.Round(1f \* gpss.a3.busy\_iterations / (gpss.a3.busy\_iterations + gpss.a3.free\_iterations),4)}\tAve.Time: {(1f \* gpss.a3.first.overall\_time + gpss.a3.second.overall\_time)/(gpss.a3.first.overall\_timestamps\_count+gpss.a3.second.overall\_timestamps\_count)}");

Console.WriteLine();

Console.WriteLine($"Q1\tEntry: {gpss.q\_a1\_entry}\tCount: {gpss.q\_a1.Count}\tMax: {gpss.q\_a1\_max}");

Console.WriteLine($"Q2\tEntry: {gpss.q\_a2\_entry}\tCount: {gpss.q\_a2.Count}\tMax: {gpss.q\_a2\_max}");

Console.WriteLine($"Q3\tEntry: {gpss.q\_a3\_entry}\tCount: {gpss.q\_a3.Count}\tMax: {gpss.q\_a3\_max}");

Console.ReadKey();

}

class GPSS

{

public GPSS(int time)

{

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

Loop(i);

}

class Generate

{

public int generate\_time;

public int main\_time;

public int additional\_time;

public void SetTime(int current\_time)

{

Random rnd = new Random();

generate\_time = current\_time + main\_time + rnd.Next(-additional\_time, additional\_time);

}

}

public class Worker

{

public struct WorkingTime

{

public int working\_time;

public int working\_main\_time;

public int working\_additional\_time;

public void SetTime(int current\_time)

{

Random rnd = new Random();

working\_time = current\_time + working\_main\_time + rnd.Next(-working\_additional\_time, working\_additional\_time);

overall\_time += working\_time - current\_time;

overall\_timestamps\_count++;

}

public int overall\_time;

public int overall\_timestamps\_count;

}

public int entries = 0;

public int busy\_iterations = 0;

public int free\_iterations = 0;

public WorkingTime first = new WorkingTime();

public WorkingTime second = new WorkingTime();

public bool is\_free = true;

}

public Worker a1 = new Worker()

{

first = new Worker.WorkingTime()

{

working\_main\_time = 12,

working\_additional\_time = 5

},

second = new Worker.WorkingTime()

{

working\_main\_time = 5,

working\_additional\_time = 2

}

};

public Worker a2 = new Worker()

{

first = new Worker.WorkingTime()

{

working\_main\_time = 15,

working\_additional\_time = 5

},

second = new Worker.WorkingTime()

{

working\_main\_time = 18,

working\_additional\_time = 3

}

};

public Worker a3 = new Worker()

{

first = new Worker.WorkingTime()

{

working\_main\_time = 20,

working\_additional\_time = 4

},

second = new Worker.WorkingTime()

{

working\_main\_time = 10,

working\_additional\_time = 3

}

};

Generate first\_gen = new Generate()

{

main\_time = 30,

additional\_time = 10

};

Generate second\_gen = new Generate()

{

main\_time = 15,

additional\_time = 3

};

public Queue<string> q\_a1 = new Queue<string>();

public Queue<string> q\_a2 = new Queue<string>();

public Queue<string> q\_a3 = new Queue<string>();

public int q\_a1\_max = 0;

public int q\_a2\_max = 0;

public int q\_a3\_max = 0;

public int q\_a1\_entry = 0;

public int q\_a2\_entry = 0;

public int q\_a3\_entry = 0;

void FirstSegment(int time)

{

if(first\_gen.generate\_time == time)

{

first\_gen.SetTime(time);

q\_a2.Enqueue("");

q\_a2\_entry++;

if (q\_a2\_max < q\_a2.Count)

q\_a2\_max = q\_a2.Count;

}

//1

if (a2.is\_free && q\_a2.Count != 0)

{

a2.is\_free = false;

a2.first.SetTime(time);

q\_a2.Dequeue();

a2.entries++;

}

else if (!a2.is\_free && a2.first.working\_time == time)

{

//q\_a2.Dequeue();

q\_a1.Enqueue("");

q\_a1\_entry++;

if (q\_a1\_max < q\_a1.Count)

q\_a1\_max = q\_a1.Count;

a2.is\_free = true;

}

//1

//2

if (a1.is\_free && q\_a1.Count != 0)

{

q\_a1.Dequeue();

a1.is\_free = false;

a1.first.SetTime(time);

a1.entries++;

}

else if (!a1.is\_free && a1.first.working\_time == time)

{

//q\_a1.Dequeue();

q\_a3.Enqueue("");

q\_a3\_entry++;

if (q\_a3\_max < q\_a3.Count)

q\_a3\_max = q\_a3.Count;

a1.is\_free = true;

}

//2

//3

if (a3.is\_free && q\_a3.Count != 0)

{

q\_a3.Dequeue();

a3.is\_free = false;

a3.first.SetTime(time);

a3.entries++;

}

else if (!a3.is\_free && a3.first.working\_time == time)

{

//q\_a3.Dequeue();

a3.is\_free = true;

first\_done++;

}

//3

}

public int first\_done = 0;

public int second\_done = 0;

void SecondSegment(int time)

{

if (second\_gen.generate\_time == time)

{

second\_gen.SetTime(time);

q\_a1.Enqueue("");

q\_a1\_entry++;

if (q\_a1\_max < q\_a1.Count)

q\_a1\_max = q\_a1.Count;

}

//1

if (a1.is\_free && q\_a1.Count != 0)

{

a1.is\_free = false;

a1.second.SetTime(time);

q\_a1.Dequeue();

a1.entries++;

}

else if (!a1.is\_free && a1.second.working\_time == time)

{

//q\_a1.Dequeue();

q\_a3.Enqueue("");

q\_a3\_entry++;

if (q\_a3\_max < q\_a3.Count)

q\_a3\_max = q\_a3.Count;

a1.is\_free = true;

}

//1

//2

if (a3.is\_free && q\_a3.Count != 0)

{

q\_a3.Dequeue();

a3.is\_free = false;

a3.second.SetTime(time);

a3.entries++;

}

else if (!a3.is\_free && a3.second.working\_time == time)

{

//q\_a3.Dequeue();

q\_a2.Enqueue("");

q\_a2\_entry++;

if (q\_a2\_max < q\_a2.Count)

q\_a2\_max = q\_a2.Count;

a3.is\_free = true;

}

//2

//3

if (a2.is\_free && q\_a2.Count != 0)

{

q\_a2.Dequeue();

a2.is\_free = false;

a2.second.SetTime(time);

a2.entries++;

}

else if (!a2.is\_free && a2.second.working\_time == time)

{

//q\_a2.Dequeue();

a2.is\_free = true;

second\_done++;

}

//3

}

void Loop(int time)

{

FirstSegment(time);

SecondSegment(time);

if (a1.is\_free) a1.free\_iterations++;

else a1.busy\_iterations++;

if (a2.is\_free) a2.free\_iterations++;

else a2.busy\_iterations++;

if (a3.is\_free) a3.free\_iterations++;

else a3.busy\_iterations++;

}

}

}

}