Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра программного обеспечения информационных технологий

ОТЧЁТ

по лабораторной работе №3

по курсу «Системный анализ и машинное моделирование»

Вариант 19

|  |  |
| --- | --- |
| Выполнил:  студент гр. 351002  Прокопович А.Ю. | Проверил:  ст. преподаватель кафедры ПОИТ  Мельник Н.И. |

Минск 2016

# **ЗАДАНИЕ**

2

π

π

2

Построить граф состояний P-схемы, представленной на рисунке ниже:

По графу составить аналитическую модель и определить вероятности состояний. Рассчитать теоретические значения показателей Q – относительная пропускная способность, Lоч – средняя длина очереди,А – абсолютная пропускная способность.

Значения параметров в соответствии с вариантом:

* π1 = 0,3;
* π2 = 0,5.

1. **ГРАФ СОСТОЯНИЙ**

В графе состояний СМО используется следующая кодировка:

*t1 p1 j p2*,

где:

*t1* – время до следующей заявки (принимает значения 1 – новая заявка будет доступна на следующем такте, 2-новая заявка доступна на текущем такте).

*p1* – текущее состояние первого канала (принимает значения 0 – канал свободен, 1 –канал обрабатывает заявку, 3 – канал занят).

*j* – количество заявок в буфере (принимает значения 1, 2).

*p2* - текущее состояние второго канала (принимает значения 0 – канал свободен, 1 – канала занят).

Также введем дополнительное обозначение:

q1 = 1- π1

q2 = 1- π2

Граф состояний представлен на рисунке ниже:

1. **АНАЛИТИЧЕСКАЯ МОДЕЛЬ**

Для нахождения вероятностей состояний построим систему алгебраических уравнений по графу, исключим одно из уравнений в системе и дополним ее нормировочным уравнением:

1. P2100 = P1 = π1\*P2 + q2\*P3 + π1\*q2\*P5
2. P1100 = P2 = π1\*P1 + π1\*q2\*P4
3. P1001 = P3 = q1\*P1 + q1\*q2\*P4
4. P2101 = P4 = q1\*P2 + π2\*P3 + π1\*π2\*P5 + q1\*q2\*P5 + q2\*P6 + π1\*q2\*P9
5. P1101 = P5 = π1\*π2\*P4 + π1\*q2\*P7
6. P1011=P6 = q1\*π2\*P4 + q1\*q2\*P7
7. P2111=P7 = q1\*π2\*P5 + π2\*P6 + q2\*P8 + π1\*π2\*P9 + q1\*q2\*P9 + π1\*q2\*P12
8. P1021=P8 = q1\*π2\*P7 + q1\*q2\*P10 + q2\*P13
9. P1111=P9 = π1\*π2\*P7 + π1\*q2\*P10
10. P2121=P10 = π2\*P8 + q1\*π2\*P9 + q2\*P11 + π1\*π2\*P12 + q1\*q2\*P12
11. P1321=P11 = q1\*π2\*P10 + π2\*P13
12. P1121=P12 = π1\*π2\*P10
13. P2321=P13 = π2\*P11 + q1\*π2\*P12



Решив данную систему уравнений с помощью Mathcad, получим значения вероятностей состояний при *π1* = *0,3* и *π2* = *0,5*:

1. P2100=P1 = 0.08295
2. P1100=P2 = 0.05178
3. P1001=P3 = 0.12083
4. P2101=P4 = 0.17931
5. P1101=P5 = 0.046712
6. P1011=P6 = 0.10899
7. P2111=P7 = 0.13210
8. P1021=P8 = 0.08689
9. P1111=P9 = 0.031972
10. P2121=P10 = 0.081043
11. P1321=P11 = 0.04065
12. P1121=P12 = 0.012156
13. P2321=P13 = 0.02458
14. **ХАРАКТЕРИСТИКИ СМО**

Требуемые характеристики СМО можно рассчитать следующим образом:

1. *Q* (относительная пропускная способность)

*Q* - вероятность успешного обслуживания заявки системой.

Заявка будет успешна обслужена в следующих случаях:

1. Первый канал свободен:

*=* P1001 +P1011+P1021 = 0.12083 + 0.10899 + 0.08689 = 0.317

1. Первый канал освободится на такте, когда приходит заявка:

*=* q1\*(P1100+P1101+P1111) = 0.7\*(0.05178 + 0.046712 + 0.031972) = 0.091

1. Первый канал не будет разблокирован на такте, когда приходит заявка:

*Qc= (1-*q1\*π2)\*P1121+q2\*P1321 = (1-0.7\*0.5)\*0.012156 + 0.5\*0.04065 = 0.0283

Тогда общая вероятность равна: Q = Qa + Qb + Qc = 0.436.

1. *Lоч* (средняя длина очереди)

*Lоч1* = 0.5

= (P1011 + P2111 + P1111) + 2\*(P1021 + P2121 + P1321 + P1121 + P2321) = 0.10899 + 0.13210 + 0.031972 + 2\*(0.08689 + 0.081043 + 0.04065 + 0.012156 + 0.02458) = 0.7637

1. *A* (абсолютная пропускная способность).

Может быть посчитана как A = Q \* λ, где λ – интенсивность потока (среднее число событий за ед.вр.). Так как канал выдает заявки каждые 2 такта, λ = 1/2 = 0.5.

Тогда A = Q \* λ = 0.51\*0.5 = 0.218

# **ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ**

public class Simulator : ISimulator<SystemState>

{

public double FirstPropability { get; set; }

public double SecondPropability { get; set; }

public int CurrentTact { get; set; }

public SystemState[] StateArray { get; set; }

public SystemState CurrentState { get; set; }

public Simulator(double firstPropability, double secondPropability, SystemState[] stateArray, SystemState initState)

{

this.FirstPropability = firstPropability;

this.SecondPropability = secondPropability;

this.CurrentState = initState;

this.StateArray = stateArray;

}

public void JumpNextState()

{

JumpCondition jumpCondition = new JumpCondition()

{

IsFirstAvailable = Distributor.IsNextTrue(FirstPropability),

IsSecondAvailable = Distributor.IsNextTrue(SecondPropability)

};

CurrentState = (SystemState) CurrentState.JumpConditionArray.First(e => e.Equals(jumpCondition)).JumpState;

}

}

public class Tester

{

public ISimulator<SystemState> Simulator { get; set; }

public int TactCount { get; set; }

public Tester(ISimulator<SystemState> simulator, int tactCount)

{

this.Simulator = simulator;

this.TactCount = tactCount;

}

public SystemState[] RunTest()

{

var result = new SystemState[TactCount];

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

{

result[i] = Simulator.CurrentState;

Simulator.JumpNextState();

}

return result;

}

}

public static class Distributor

{

public static bool IsNextTrue(double propability)

{

return RandomGenerator.GetNext() > propability;

}

}

public class Stats

{

SystemState[] States { get; set; }

TestStateResult[] ResultStates { get; set; }

public Stats(SystemState[] states, TestStateResult[] resultStates)

{

this.States = states;

this.ResultStates = resultStates;

}

public StatsResult GetFrequencyStats()

{

var resultStates = ResultStates.Select(e => e.State).ToArray();

var result = new List<double>();

double sum = 0;

foreach (var state in States)

{

double frequency = 1.0 \* resultStates.Count(e => e.Index == state.Index) / ResultStates.Length;

sum += frequency;

result.Add(frequency);

}

return new StatsResult()

{

FrequencyList = result,

FrequencySum = sum,

RelativeBandwidth = GetRelativeBandwidth(),

AbsoluteBandwidth = GetAbsoluteBandwidth(),

AverageBufferLength = GetAverageBufferLength()

};

}

private double GetRelativeBandwidth()

{

return 1.0\*ResultStates.Select(e => e.Condition).Count(e => e.IsSecondAvailable == true)/

ResultStates.Count();

}

private double GetAbsoluteBandwidth()

{

return GetRelativeBandwidth()/2;

}

private double GetAverageBufferLength()

{

return 1.0\*ResultStates.Select(e => e.State).Count(e => e.BufferState == 2)\*2/ResultStates.Count() +

1.0\*ResultStates.Select(e => e.State).Count(e => e.BufferState == 1)/ResultStates.Count();

}

}