Министерство образования Республики Беларусь

Учреждение Образования

«Белорусский государственный университет информатики и радиоэлектроники»

Лабораторная работа №4

Разработка, отладка и исследование программной модели непрерывно-стохастической СМО

|  |  |  |
| --- | --- | --- |
| Выполнил студент гр. 281011  Радионов М.Г. |  | Проверила  Лашкевич Е.М. |

2015

1. Исходные данные

Цель лабораторной работы: построить имитационную модель непрерывно-стохастической СМО и исследовать ее (разработать алгоритм и написать имитирующую программу, предусматривающую сбор и статистическую обработку данных для получения оценок заданных характеристик СМО).

Выполнить задание необходимо в соответствии с выданным вариантом № 21.

Одноканальная СМО с неограниченной очередью. Входной поток простейший с интенсивностью 2 заяв/час. Найти среднюю длину очереди и среднее время ожидания в очереди, если поток обслуживания имеет:

а) показательное распределение времени обслуживания, μ=2,5 заяв/час;

б) равномерное распределение времени обслуживания, а=0,2час, b=0,6час;

в) нормальное распределение времени обслуживания, μ=0.4час, σ=0,1час;

2. Анализ задания

Одноканальная СМО с неограниченной очередью – M/M/1/∞  
 - входной поток заявок – простейший с интенсивностью λ,  
 - поток обслуживаний – простейший с интенсивностью μ,  
 - количество каналов обслуживания – 1,  
 - количество мест ожидания – не ограничено  
 Финальные вероятности существуют и могут быть определены, если интенсивность  
поступления заявок ниже интенсивности их обслуживания, т.е.  
При ω=1 очередь будет бесконечно расти

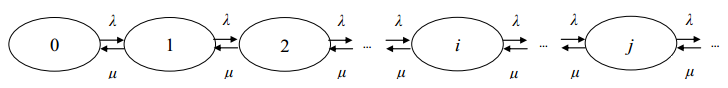


Рисунок 1.1 - ДИП для системы M/M/1/∞

3. Графическое описание алгоритма

T - моделируемое системное время;

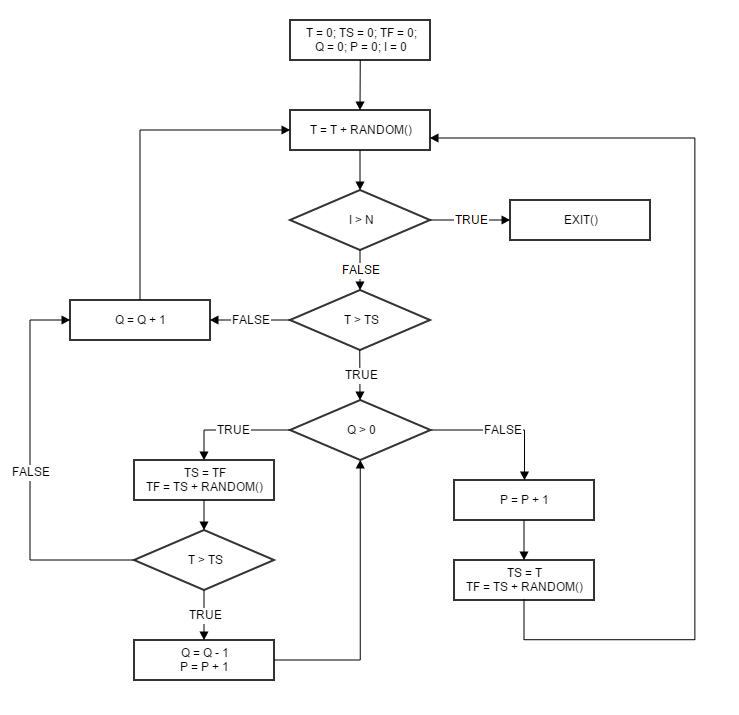
TS - время начала обработки заявки;

TF - время окончания обработки заявки;

Q - количество заявок в очереди;

P - количество обработанных заявок;

I - количетство итераций.



4. Нахождение искомых параметров

Система была запущена на 1000 итераций.

Для исходных данных: показательное распределение времени обслуживания, μ=2,5 заяв/час.  
  
 Обработано: 950

Средняя длина очереди: 0.21

Среднее время ожидания: 0.07486511686497586

Для исходных данных: равномерное распределение времени обслуживания, а=0,2час, b=0,6час.

Обработано: 306

Средняя длина очереди: 332.534

Среднее время ожидания: 155.7084051802576

Для исходных данных: нормальное распределение времени обслуживания, μ=0.4час, σ=0,1час.

Обработано: 287

Средняя длина очереди: 355.564

Среднее время ожидания: 178.07537188210205

5. Код программы

'use strict';

function sum(array) {

return array.reduce(function (a, b) {

return a + b;

}, 0);

}

function average(array) {

return sum(array) / array.length;

}

// Generator

var generator = (function () {

var x = 7,

a = 15,

c = 3,

n = 32,

i = 100 \* 20,

// i = 20,

m = Math.pow(2, n);

return {

original: original,

exponential: exponential,

uniform: uniform,

gaussian: gaussian

};

////////

function original() {

return pickRand(seq());

}

function exponential(lambda) {

return pickRand(exponentialDist(seq(), lambda));

}

function uniform(a, b) {

return pickRand(uniformDist(seq(), a, b));

}

function gaussian(mean, variance) {

return pickRand(gaussianDist(seq(), mean, variance, 1));

}

////////

function seq() {

var originalSequenceClean = originalDistClean(x, i);

var originalSequence = originalDistNormalize(originalSequenceClean);

return originalSequence;

}

function originalDistClean(xStart, k) {

var seq = [xStart];

for (var i = 0; i < k; i++) {

seq.push((a \* seq[i] + c) % m);

}

return seq;

}

function originalDistNormalize(seq) {

return seq.map(function (x) {

return x / m;

});

}

function exponentialDist(sequence, lambda) {

return sequence.map(function (r) {

return (-1 / lambda) \* log10(r);

});

}

function uniformDist(sequence, a, b) {

return sequence.map(function (r) {

return a + (b - a) \* r;

});

}

function gaussianDist(sequence, mean, variance, n) {

return sequence.map(function (r) {

var sumRands = sum(pickRands(sequence, n));

return mean + variance \* Math.sqrt(12 / n) \* (sumRands - n / 2);

});

}

////////

function pickRand(array) {

return array[Math.floor(array.length \* Math.random())];

}

function pickRands(array, n) {

var result = [];

for (var i = 0; i < n; i++) {

result.push(pickRand(array));

}

return result;

}

function log10(x) {

return Math.log(x) / Math.LN10;

}

}());

// Timeline

var timeline = (function () {

return {

run: run

};

function run(times, generate) {

var queued = [];

var processed = 0;

var systemTimeInc = 0.5;

var isInitial = true;

var system = {

time: 0,

serveFinish: 0,

serveStart: 0

};

var statistics = {

queue: [],

wait: []

};

function iteration (generate) {

statistics.queue.push(queued.length);

system.time += systemTimeInc;

if (system.time > system.serveFinish) {

if (queued.length > 0) {

do {

system.serveStart = system.serveFinish;

system.serveFinish = system.serveStart + generate();

if (system.time > system.serveFinish) {

var startTime = queued.shift();

processed += 1;

statistics.wait.push(system.serveStart - startTime);

} else {

queued.push(system.time);

break;

}

} while(queued.length > 0);

} else {

if (!isInitial) {

statistics.wait.push(0);

processed += 1;

} else {

isInitial = false;

}

system.serveStart = system.time;

system.serveFinish = system.serveStart + generate();

}

} else {

queued.push(system.time);

}

}

for (var j = 0; j < times; j++) {

iteration(generate);

}

console.log('--------------------------------------------------');

console.log('Processed', processed);

console.log('Average queue length', average(statistics.queue));

console.log('Average wait time', average(statistics.wait));

console.log('Statistics', statistics);

}

}());

function task1() {

var number = generator.exponential(2.5);

// console.log(number);

return number;

}

function task2() {

var number = generator.uniform(0.2, 0.6);

// console.log(number);

return number;

}

function task3() {

var number = generator.gaussian(0.4, 0.1);

return number;

}

var iterations = 1000;

timeline.run(iterations, task1);

timeline.run(iterations, task2);

timeline.run(iterations, task3);