НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп‘ютерних систем

**Лабораторна робота №3**

***З дисципліни: «Системне програмне забезпечення»***

«Дослідження дисциплін обслуговування заявок при обмежених ресурсах»

Студент КВ-42**:** Савицький Т.П.

Перевірив(ла):

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

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

Київ-2017

**Завдання**

1. Згідно варіанту виконати по тактову візуалізацію системи обслуговування заявок з розподіленим ресурсом, параметрами системи є діапазон ваг заявок, діапазон інтервалів виникнення наступної заявки (інтенсивність вхідного потоку), діапазон пріоритетів (для пріоритетних дисциплін обслуговування). В систему може надходити будь-яка кількість заявок.
2. Якщо система обслуговування з чергами, то кількість черг не більша 32. Якщо система обслуговування з пріоритетами, то число пріоритетів не більш ніж 32.
3. Побудувати графіки залежності середнього часу очікування від інтенсивності вхідного потоку заявок та залежність проценту простою ресурсу від інтенсивності вхідного потоку заявок.
4. Для пріоритетних систем побудувати графік залежності середнього часу очікування від пріоритету при фіксованій інтенсивності вхідного потоку заявок. Для систем без пріоритетів побудувати графік залежності кількості заявок від часу очікування при фіксованій інтенсивності вхідного потоку заявок.
5. Пояснити форму графіків.

Для графіку з усередненням часу є крива, яка схожа на степеневу функцію. Чим більше кількість черг, тим більше прилягає до вісі Х, аналогічна ситуація і з часом.

Для графіку з затримкою - чим менше кількості черг, тим менше часу потрібно для завершення черги

Для графіка з часом очікуванння - аналогічна ситуацій як і з графіком затримки.

**Текст програми**

//

// appController.js

//

import Observer from './common/observer.js';

import Variables from './common/variables.js'

import ModelController from './modelController.js';

import ViewController from './viewController.js';

export class AppController {

constructor() {

this.model = new ModelController();

this.view = new ViewController(this.model);

Observer.addListener(Variables.isFreeTaskFIFO, this.isFreeProcessFIFO.bind(this));

Observer.addListener(Variables.tasksIsDone, this.tasksIsDone.bind(this));

this.model.runProcess(this.model.queues);

}

isFreeProcessFIFO(task) {

console.log(task);

}

tasksIsDone(queues, process) {

console.log(queues)

this.view.taskIsDone(queues, process);

}

}

//

// modelController.js

//

import Variables from './common/variables.js';

import Observer from './common/observer.js';

import Process from './Model/process.js';

import Task from './Model/task.js';

export default class ModelController {

constructor() {

this.time = 1000;

this.processor = new Process('Single Process');

this.queues = [{

name: '1',

tasks: [

new Task('1', 200),

new Task('2', 150),

new Task('2', 150)]

}, {

name: '2',

tasks: [

new Task('1', 100),

new Task('2', 50),

new Task('3', 200)]

}, {

name: '3',

tasks: [

new Task('1', 50),

new Task('2', 25),

new Task('3', 100)]

}, {

name: '4',

tasks: [

new Task('1', 55),

new Task('2', 65),

new Task('3', 40)]

}, {

name: '5',

tasks: [

new Task('1', 80),

new Task('2', 35),

new Task('3', 40)]

}, {

name: '6',

tasks: [

new Task('2', 75),

new Task('3', 100)]

}, {

name: '7',

tasks: [

new Task('1', 150)]

}, {

name: '8',

tasks: [

new Task('1', 50),

new Task('2', 250)]

}, {

name: '9',

tasks: [

new Task('1', 75),

new Task('2', 25),

new Task('3', 10)]

}, {

name: '10',

tasks: [

new Task('1', 20),

new Task('3', 10)]

}];

}

runProcess(queues) {

let counterDone = 0;

this.processor.beginTime = performance.now();

let intervalTasks = setInterval(() => {

if (!this.processor.isBusy) {

this.\_carriage(queues[counterDone], this.processor);

counterDone++;

}

if (counterDone === queues.length) {

this.processor.finishTime = performance.now();

clearInterval(intervalTasks);

Observer.emit(Variables.tasksIsDone, queues, this.processor);

}

}, 0);

}

\_carriage(queue, process) {

queue.beginTime = performance.now();

queue.delayTime = queue.beginTime - process.beginTime;

for (let task of queue.tasks) {

task.beginTime = performance.now();

process.makingFIFO(task);

task.done = true;

}

queue.finishTime = performance.now();

queue.totalTime = queue.finishTime - queue.beginTime;

queue.isDone = true;

Observer.emit(Variables.isFreeTaskFIFO, queue.tasks);

}

}

//

// viewController.js

//

export default class ViewController {

constructor(model) {

this.model = model;

}

taskIsDone(queues, process) {

let data = [];

// for average time

let averageTime = 0;

for (let queue of queues) {

averageTime += queue.delayTime;

}

averageTime /= queues.length;

for (let i = queues.length; i > 0; i--) {

data.push({'x': averageTime / i, 'y': i});

}

this.\_generateSVG([data], 0, averageTime + 100, queues.length + 1, '"Average time" >> Time (ms)');

// for percent data

data = [];

const deltaTime = process.finishTime - process.beginTime;

for (let i = queues.length; i > 0; i--) {

data.push({'x': (queues[i-1].delayTime / deltaTime) \* 100, 'y': i});

}

this.\_generateSVG([data], 0, 100, queues.length + 1, '"t(delay)/t(total)" >> %');

// for waiting time

data = [];

for (let i = queues.length; i > 0; i--) {

data.push({'x': queues[i-1].delayTime, 'y': i});

}

this.\_generateSVG([data], 0, queues[queues.length-1].delayTime + 10, queues.length + 1, '"Waiting time" >> Time (ms)');

}

\_generateSVG(data, startX, finishX, lengthY, textX) {

const colors = [];

for (let i = 0; i < data.length; i++) {

colors.push(this.\_getRandomColor());

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Create Margins and Axis and hook our zoom function

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

const margin = {top: 20, right: 30, bottom: 50, left: 50},

width = window.innerWidth - margin.left - margin.right \* 2,

height = 250 - margin.top - margin.bottom;

const x = d3.scale.linear()

.domain([startX, finishX])

.range([0, width]);

const y = d3.scale.linear()

.domain([0, lengthY])

.range([height, 0]);

const xAxis = d3.svg.axis()

.scale(x)

.tickSize(-height)

.tickPadding(10)

.tickSubdivide(true)

.orient("bottom");

const yAxis = d3.svg.axis()

.scale(y)

.tickPadding(10)

.tickSize(-width)

.tickSubdivide(true)

.orient("left");

const svg = d3.select("body").append("svg")

.attr("width", width + margin.left + margin.right)

.attr("height", height + margin.top + margin.bottom)

.append("g")

.attr("transform", "translate(" + margin.left + "," + margin.top + ")");

svg.append("g")

.attr("class", "x axis")

.attr("transform", "translate(0," + height + ")")

.call(xAxis);

svg.append("g")

.attr("class", "y axis")

.call(yAxis);

svg.append("g")

.attr("class", "y axis")

.append("text")

.attr("class", "axis-label")

.attr("transform", "rotate(-90)")

.attr("y", (-margin.left) + 10)

.attr("x", -height/2)

.text('# queue');

svg.append("g")

.attr("class", "x axis")

.append("text")

.attr("class", "axis-label")

.attr("y", (height + margin.top) + 25)

.attr("x", width/2)

.text(textX);

svg.append("clipPath")

.attr("id", "clip")

.append("rect")

.attr("width", width)

.attr("height", height);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Create D3 line object and draw data on our SVG object

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

const line = d3.svg.line()

.interpolate("linear")

.x((d) => x(d.x))

.y((d) => y(d.y));

svg.selectAll('.line')

.data(data)

.enter()

.append("path")

.attr("class", "line")

.attr("clip-path", "url(#clip)")

.attr('stroke', () => this.\_getRandomColor())

.attr("d", line);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Draw points on SVG object based on the data given

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

const points = svg.selectAll('.dots')

.data(data)

.enter()

.append("g")

.attr("class", "dots")

.attr("clip-path", "url(#clip)");

points.selectAll('.dot')

.data((d, index) => {

const a = [];

d.forEach((point) => {

a.push({'index': index, 'point': point});

});

return a;

})

.enter()

.append('circle')

.attr('class','dot')

.attr("r", 2.5)

.attr('fill', (d,) => {

return colors[d.index%colors.length];

})

.attr("transform", (d) =>

"translate(" + x(d.point.x) + "," + y(d.point.y) + ")"

);

}

\_getRandomColor() {

const letters = '0123456789ABC';

let color = '#';

for (let i = 0; i < 6; i++) {

color += letters[Math.floor(Math.random() \* 13)];

}

return color;

}

}

//

// process.js

//

export default class Process {

constructor(title, performance = 100) {

this.title = title;

this.performance = performance;

this.\_isBusy = false;

}

get isBusy() {

return this.\_isBusy;

}

makingFIFO(task) {

this.\_isBusy = true;

this.\_delay(task);

this.\_isBusy = false;

}

\_delay(task) {

task.runningTime = task.time;

task.startTime = performance.now();

const finishTime = performance.now() + task.runningTime;

while (performance.now() <= finishTime) {}

task.finishTime = performance.now();

task.delayTime = Math.abs(task.runningTime - (task.finishTime - task.beginTime));

task.totalTime = task.finishTime - task.beginTime;

}

}

//

// task.js

//

export default class Task {

constructor(title, time, priority) {

this.title = title;

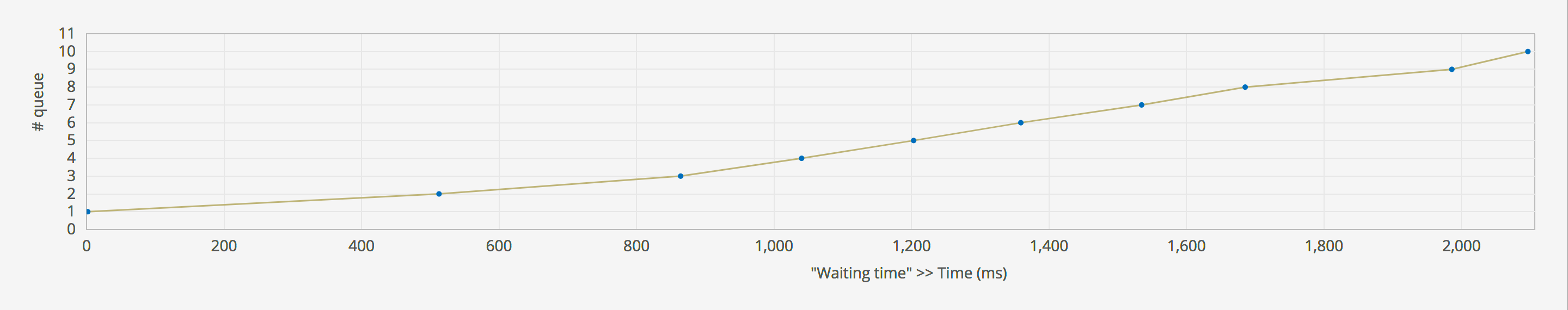
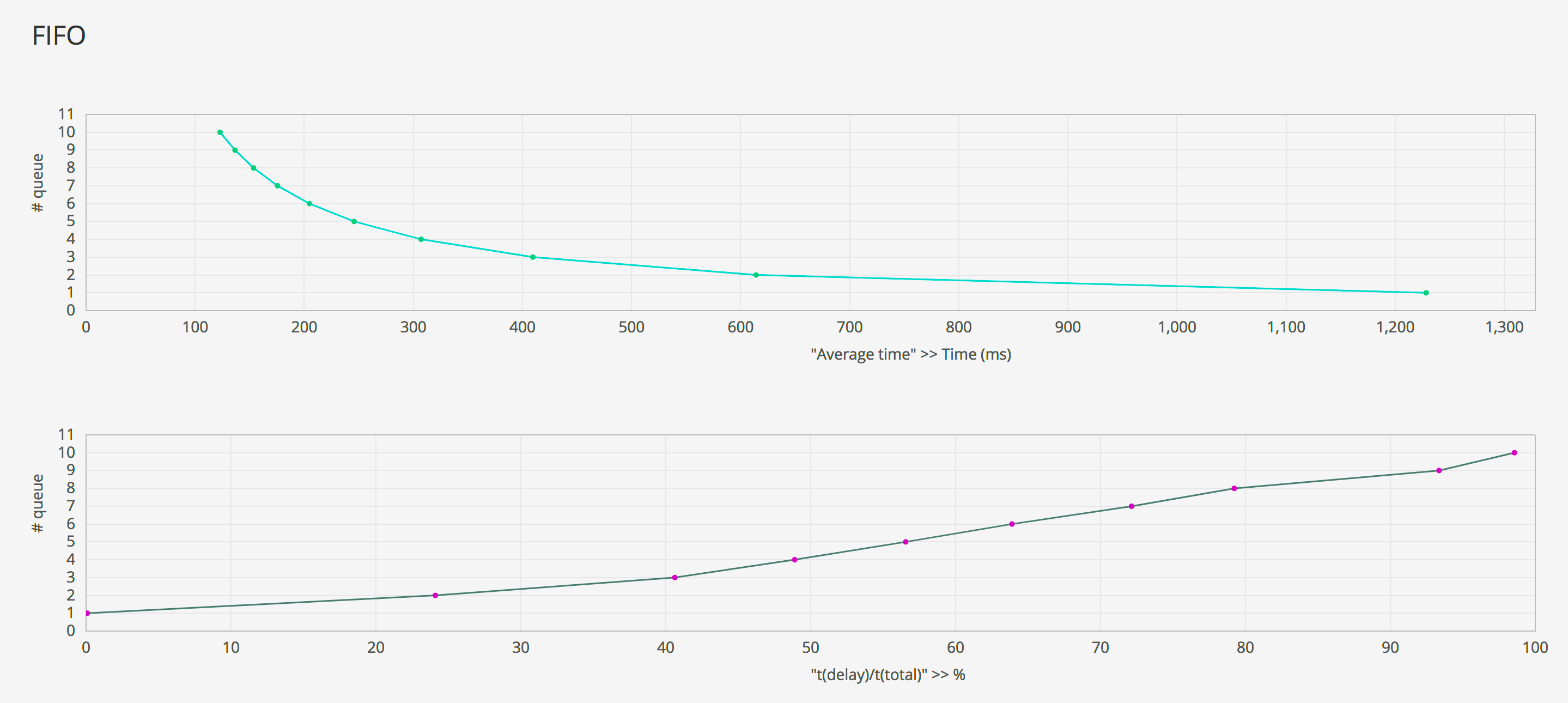
this.time = time;

this.done = false;

this.durationTime = 0;

}

}

**Тести**