

Міністерство освіти і науки України

Національний технічний університет України «КПІ

імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

**ЗВІТ**

лабораторної роботи №2

з дисципліни «Моделювання систем»

Перевірила:

Дифучина О. Ю.

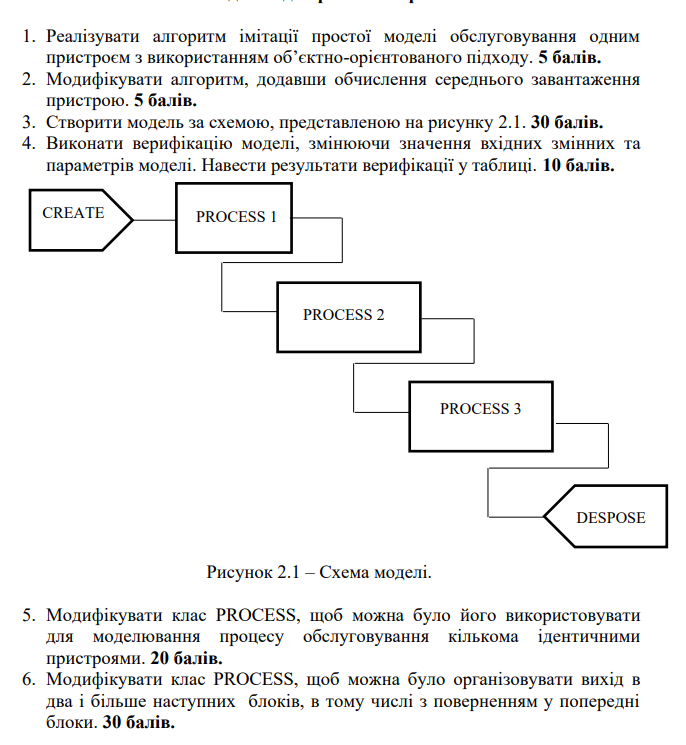
Виконав:

Студент Гр. ІП-01

Пашковський Є. С.

Київ 2023

Завдання



Хід роботи

Код для виконання завдання:

import Create from './Create';

import Distribution from './Distribution';

import Model from './Model';

import Process from './Process';

const create = new Create(2);

const p1 = new Process(1);

const p2 = new Process(1);

const p3 = new Process(1);

create.setNextElements([{ element: p1, probability: 1 }]);

p1.setNextElements([{ element: p2, probability: 1 }]);

p2.setNextElements([{ element: p3, probability: 1 }]);

p1.setMaxQueueLength(2);

p2.setMaxQueueLength(2);

p3.setMaxQueueLength(3);

create.setDistribution(Distribution.EXPONENTIAL);

p1.setDistribution(Distribution.EXPONENTIAL);

p2.setDistribution(Distribution.EXPONENTIAL);

p3.setDistribution(Distribution.EXPONENTIAL);

p1.setResourcesCount(1);

p2.setResourcesCount(1);

p3.setResourcesCount(1);

const model = new Model([create, p1, p2, p3]);

model.simulate(1000);

import CustomRandom from './CustomRandom';

import Distribution from './Distribution';

export default abstract class Element {

  protected static nextId = 0;

  public readonly id: number;

  public readonly name: string;

  private nextElements: { element: Element; probability: number }[] = [];

  private nextT: number = 0;

  private currentT: number = 0;

  private distribution: Distribution;

  public readonly delayMean: number;

  public readonly delayVariance: number;

  protected quantity: number = 0;

  protected state: number = 0;

  constructor(

    name = '',

    distribution = Distribution.STATIC,

    delayMean = 0,

    delayVariance = 0

  ) {

    this.id = Element.nextId++;

    this.name = name || 'element' + this.id;

    this.distribution = distribution;

    this.delayMean = delayMean;

    this.delayVariance = delayVariance;

  }

  public getDelay(): number {

    switch (this.distribution) {

      case Distribution.NORMAL:

        return CustomRandom.generateNormal(

          Math.sqrt(this.delayVariance),

          this.delayMean

        );

      case Distribution.EXPONENTIAL:

        return CustomRandom.generateExponential(1 / this.delayMean);

      case Distribution.UNIFORM:

        return CustomRandom.generateUniform() \* 2 \* this.delayMean;

      case Distribution.STATIC:

      default:

        return this.delayMean;

    }

  }

  public inAction() {}

  public outAction() {

    this.quantity++;

  }

  public getQuantity() {

    return this.quantity;

  }

  public setNextElements(

    elements: { element: Element; probability: number }[]

  ) {

    if (elements.reduce((acc, el) => acc + el.probability, 0) !== 1)

      throw new Error('Sum of possibilities should equal 1');

    this.nextElements = elements;

  }

  public getNextElement() {

    const rand = Math.random();

    let sum = 0;

    for (const { element, probability } of this.nextElements) {

      sum += probability;

      if (rand < sum) return element;

    }

    return undefined;

  }

  public getNextT() {

    return this.nextT;

  }

  public setNextT(t: number) {

    this.nextT = t;

  }

  public getCurrentT() {

    return this.currentT;

  }

  public setCurrentT(t: number) {

    this.currentT = t;

  }

  public doStatistics(delta: number) {}

  public printResult() {

    console.log(this.name + ' quantity = ' + this.quantity + '\n');

  }

  public printInfo() {

    console.log(

      this.name +

        ' state= ' +

        this.state +

        ' quantity = ' +

        this.quantity +

        ' tnext= ' +

        this.nextT

    );

  }

  public setDistribution(distribution: Distribution) {

    this.distribution = distribution;

  }

}

enum Distribution {

  EXPONENTIAL = 'exponential',

  NORMAL = 'normal',

  UNIFORM = 'uniform',

  STATIC = 'static',

}

export default Distribution;

import generateRandomOne from '../../lab1/src/server/generateRandomOne';

import generateRandomThree from '../../lab1/src/server/generateRandomThree';

import generateRandomTwo from '../../lab1/src/server/generateRandomTwo';

export default class CustomRandom {

  public static generateNormal(o: number, a: number) {

    return generateRandomTwo(o, a);

  }

  public static generateExponential(lambda: number) {

    return generateRandomOne(lambda);

  }

  public static generateUniform() {

    return generateRandomThree();

  }

}

import Distribution from './Distribution';

import Element from './Element';

export default class Create extends Element {

  constructor(delay: number) {

    super('create', Distribution.STATIC, delay);

  }

  public outAction() {

    super.outAction();

    super.setNextT(super.getCurrentT() + super.getDelay());

    super.getNextElement()?.inAction();

  }

}

import Element from './Element';

import Process from './Process';

export default class Model {

  private elements: Element[] = [];

  private currentT: number = 0;

  private nextT: number = 0;

  private event: number = 0;

  constructor(elements: Element[]) {

    this.elements = elements;

  }

  public simulate(t: number) {

    while (this.currentT < t) {

      this.nextT = Infinity;

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

        const element = this.elements[i];

        if (element.getNextT() < this.nextT) {

          this.nextT = element.getNextT();

          this.event = i;

        }

      }

      console.log(

        '\nIts time for event in ' +

          this.elements[this.event].name +

          ', time = ' +

          this.nextT

      );

      for (const element of this.elements) {

        element.doStatistics(this.nextT - this.currentT);

      }

      this.currentT = this.nextT;

      for (const element of this.elements) {

        element.setCurrentT(this.currentT);

      }

      this.elements[this.event].outAction();

      for (const element of this.elements) {

        if (element.getNextT() === this.currentT) {

          element.outAction();

        }

      }

      this.printInfo();

    }

    this.printResult();

  }

  public printInfo() {

    for (const element of this.elements) {

      element.printInfo();

    }

  }

  public printResult() {

    console.log('\n-------------RESULTS-------------');

    for (const element of this.elements) {

      element.printResult();

      if (element instanceof Process) {

        console.log(

          `mean length of queue = ${

            element.getMeanQueue() / this.currentT

          }\nmean resources busy = ${

            element.getMeanBusyResources() / this.currentT

          }\nfailure probability = ${

            element.getFails() / (element.getQuantity() + element.getFails())

          }\n`

        );

      }

    }

  }

}

import Distribution from './Distribution';

import Element from './Element';

export default class Process extends Element {

  private queue = 0;

  private maxQueueLength = Number.MAX\_VALUE;

  private fails = 0;

  private meanQueue = 0;

  private meanBusyResources = 0;

  private resourcesCount = 1;

  constructor(delay: number) {

    super('process' + Element.nextId, Distribution.STATIC, delay);

  }

  public inAction() {

    super.inAction();

    if (this.state !== this.resourcesCount) {

      this.state++;

      this.setNextT(this.getCurrentT() + this.getDelay());

      return;

    }

    if (this.queue < this.maxQueueLength) {

      this.queue++;

      return;

    }

    this.fails++;

  }

  public outAction() {

    const nextElement = super.getNextElement();

    this.quantity += this.state;

    for (let i = 0; i < this.state; i++) {

      nextElement?.inAction();

    }

    this.setNextT(Infinity);

    this.state = 0;

    if (this.queue > 0) {

      while (this.state < this.resourcesCount && this.queue > 0) {

        this.queue--;

        this.state++;

      }

      this.setNextT(this.getCurrentT() + this.getDelay());

    }

  }

  public doStatistics(delta: number) {

    this.meanQueue += this.queue \* delta;

    this.meanBusyResources += this.state \* delta;

  }

  public getQueue() {

    return this.queue;

  }

  public getMeanQueue() {

    return this.meanQueue;

  }

  public getMeanBusyResources() {

    return this.meanBusyResources;

  }

  public setMaxQueueLength(length: number) {

    this.maxQueueLength = length;

  }

  public setResourcesCount(resourcesCount: number) {

    this.resourcesCount = resourcesCount;

  }

  public getFails() {

    return this.fails;

  }

  public printInfo() {

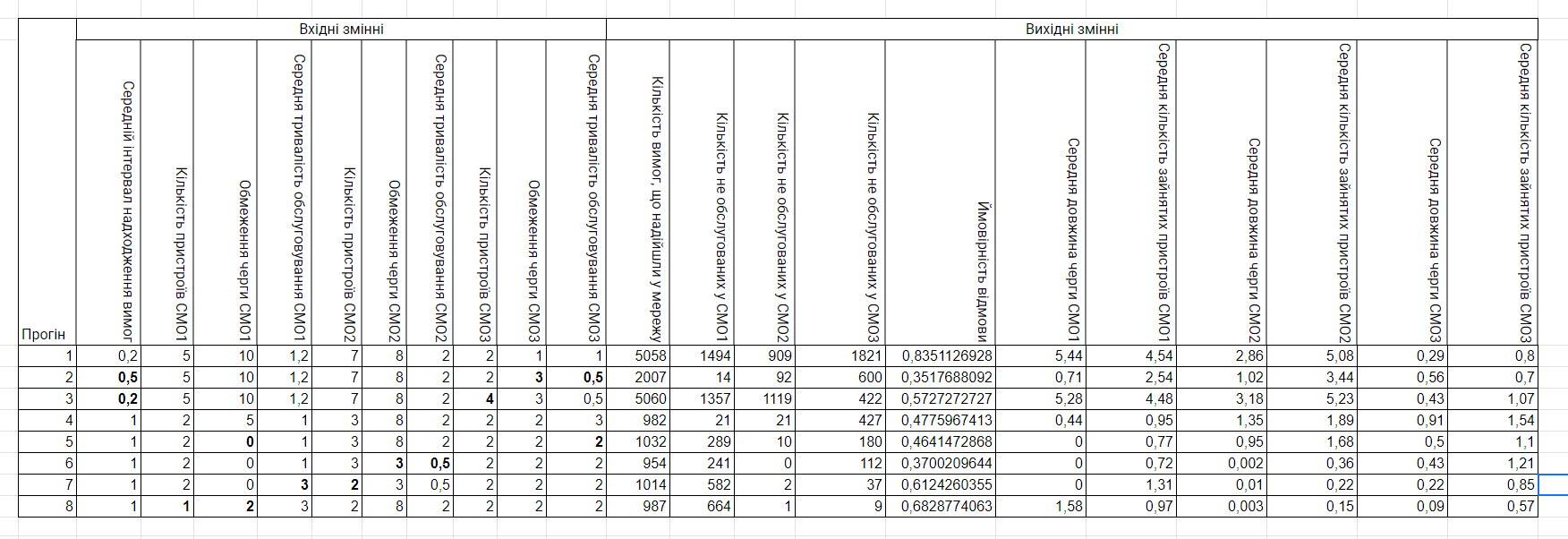
    super.printInfo();

    console.log('fails = ' + this.fails);

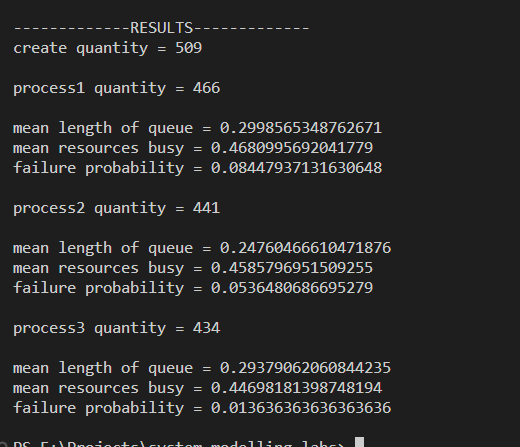
  }

}

Верифікація моделі:



Результати роботи коду:



Висновки: під час виконання цього завдання було побудовано дискретно-імітаційну модель та проведено її дослідження та верифікацію.