НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра обчислювальної техніки
(повна назва кафедри, циклової комісії)

РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА

з дисципліни «Інтелектуальні вбудовані системи»
(назва дисципліни)

на тему: «Дослідження роботи планувальників роботи систем реального часу»

Студентки 3 курсу групи ІП-83
спеціальності
121 «Інженерія програмного забезпечення»

Мазур С.В.
(прізвище та ініціали)

Керівник доцент Волокіта А.Н.

Київ – 2021 рік

Дослідження роботи планувальників роботи систем реального часу

Мета роботи - змоделювати роботу планувальника задач у системі реального часу

Основні теоретичні відомості

Планування виконання завдань є однією з ключових концепцій в багатозадачності і багатопроцесорних систем, як в операційних системах загального призначення, так і в операційних системах реального часу. Планування полягає в призначенні пріоритетів процесам в черзі з пріоритетами.

Найважливішою метою планування завдань є якнайповніше завантаження доступних ресурсів. Для забезпечення загальної продуктивності системи планувальник має опиратися на:

- Використання процесора(-ів) — дати завдання процесору, якщо це можливо.
- Пропускна здатність — кількість процесів, що виконуються за одиницю часу.
- Час на завдання — кількість часу, для повного виконання певного процесу.
- Очікування — кількість часу, який процес очікує в черзі готових.
- Час відповіді — час, який проходить від подання запиту до першої відповіді на запит.
- Справедливість — Рівність процесорного часу для кожної нитi

У середовищах обчислень реального часу, наприклад, на пристроях, призначених для автоматичного управління в промисловості (наприклад, робототехніка), планувальник завдань повинен забезпечити виконання процесів в перебігу заданих часових проміжків (час відгуку); це критично для підтримки коректної роботи системи реального часу.

Завдання на лабораторну роботу

1. Змоделювати планувальник роботи системи реального часу. Дві дисципліни планування: перша – RR, друга задається викладачем або обирається самостійно.
2. Знайти наступні значення:
    1) середній розмір вхідної черги заявок, та додаткових черг (за їх наявності);
    2) середній час очікування заявки в черзі;
    3) кількість прострочених заявок та її відношення до загальної кількості заявок
3. Побудувати наступні графіки:
    1) Графік залежності кількості заявок від часу очікування при фіксованій інтенсивності вхідного потоку заявок.
    2) Графік залежності середнього часу очікування від інтенсивності вхідного потоку заявок.
    3) Графік залежності проценту простою ресурсу від інтенсивності вхідного потоку заявок

Лістинг програми

```
'use strict';

const colorize = (colorId) => (str) =>
  `\x1b[${100 + (colorId % 7)}m\x1b[97m${str}\x1b[0m`;

class Task {
  constructor(tStart, tCalc, tDeadline, that = null) {
    Object.assign(this, { tStart, tCalc, tDeadline, that });
  }

  getInstance = (index, wasStart) => {
    const { tStart, tCalc, tDeadline } = this;
    const newTask = new Task(tStart, tCalc, tDeadline, this);
    newTask.counter = index;
    newTask.rest = tCalc;
    newTask.wasStart = wasStart;
    return newTask;
  };

  getProto = () => this.that;

  toString = () => JSON.stringify(this);
}

class SchedulingAlgorithm {
  workTime = 0;
  queue = [];
  history = [];
  counter = 0;
  isLogging = false;

  someArr = new Array(1000).fill(null).map(() => []); //
  counter1 = 0; //
  counter2 = 0; //

  getSomeArr = () => this.someArr; //

  constructor(tasksArr) {
    this.tasksArr = tasksArr;
  }

  check = (changedIndex) => {
    const { workTime, tasksArr, isLogging } = this;
    if (changedIndex !== null) {
```

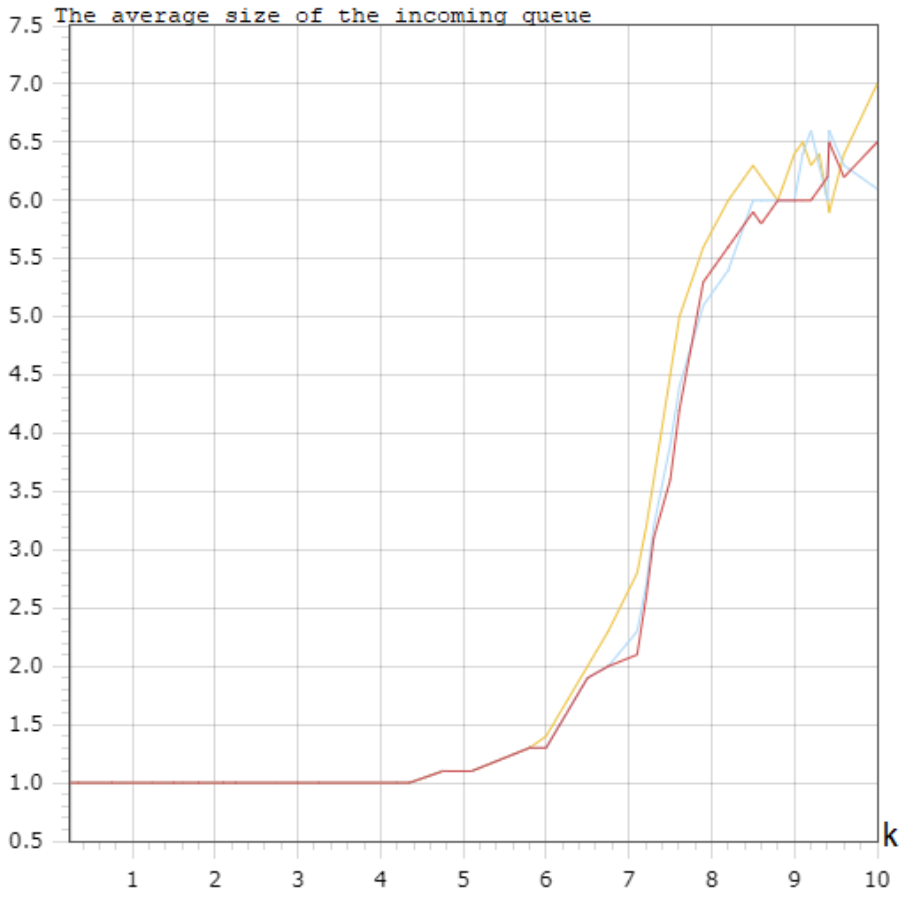
```
      this.queue[changedIndex].rest--;
      this.history.push({
        when: workTime,
        task: this.queue[changedIndex].getProto(),
        flag: 'changed',
      });
    }
    this.queue = this.queue.filter((task) => {
      if (task.rest < 1) {
        this.someArr[workTime].push(workTime - task.wasStart); //
        this.counter2++; //
        this.history.push({
          when: workTime,
          task: task.getProto(),
          flag: 'completed',
        });
        isLogging &&
          console.log(
            `\x1b[34m WorkTime: ${workTime}. Task ${task.toString()} completed.
\x1b[0m`
          );
        return false;
      }
      if (workTime + task.rest > task.wasStart + task.tDeadline) {
        this.history.push({
          when: workTime,
          task: task.getProto(),
          flag: 'overdue',
        });
        isLogging &&
          console.log(
            `\x1b[31m WorkTime: ${workTime}. Task ${task.toString()} deadline was
overdue.\x1b[0m`
          );
        return false;
      }
      return true;
    });
    tasksArr.forEach((task, i, arr) => {
      if (workTime % task.tStart === 0) {
        const newTask = task.getInstance(this.counter, workTime);
        this.history.push({
          when: workTime,
          task: arr[i],
          flag: 'added',
```

```
      });
      this.counter++;
      this.queue.push(newTask);
    }
  });
};

distribute = () => {};

iterate = () => {
  const { check, distribute, isLogging } = this;
  check(distribute());
  isLogging && console.log(this.queue.map((task) => task.toString()));
  this.workTime++;
  return this;
};

toString = () => {
  const { tasksArr, history, workTime } = this;
  let result = '';
  const historyArray = new Array(tasksArr.length).fill(null).map(() => []);
  history.forEach((event) => {
    const index = tasksArr.findIndex((task) => task === event.task);
    if (index !== -1) {
      const prevValue = historyArray[index][event.when];
      if (prevValue) {
        const prevRank = flags.find((flag) => flag.name === prevValue).rank;
        const curRank = flags.find((flag) => flag.name === event.flag).rank;
        historyArray[index][event.when] =
          prevRank < curRank ? prevValue : event.flag;
      } else historyArray[index][event.when] = event.flag;
    }
  });
  historyArray.forEach((task, taskNumber) => {
    result += 'Task ' + taskNumber + '\n';
    const taskColorized = colorize(taskNumber + 2);
    for (let i = 0; i < task.length; i++) {
      const foundFlag = flags.find((flag) => flag.name === task[i]);
      result += foundFlag ? taskColorized(foundFlag.symbol) : ' ';
    }
    result += '\n';
  });
  result += 'All tasks\n';
  for (let i = 0; i < workTime; i++) {
    let prevFlag = flags.find((flag) => flag.name === 'default');
```

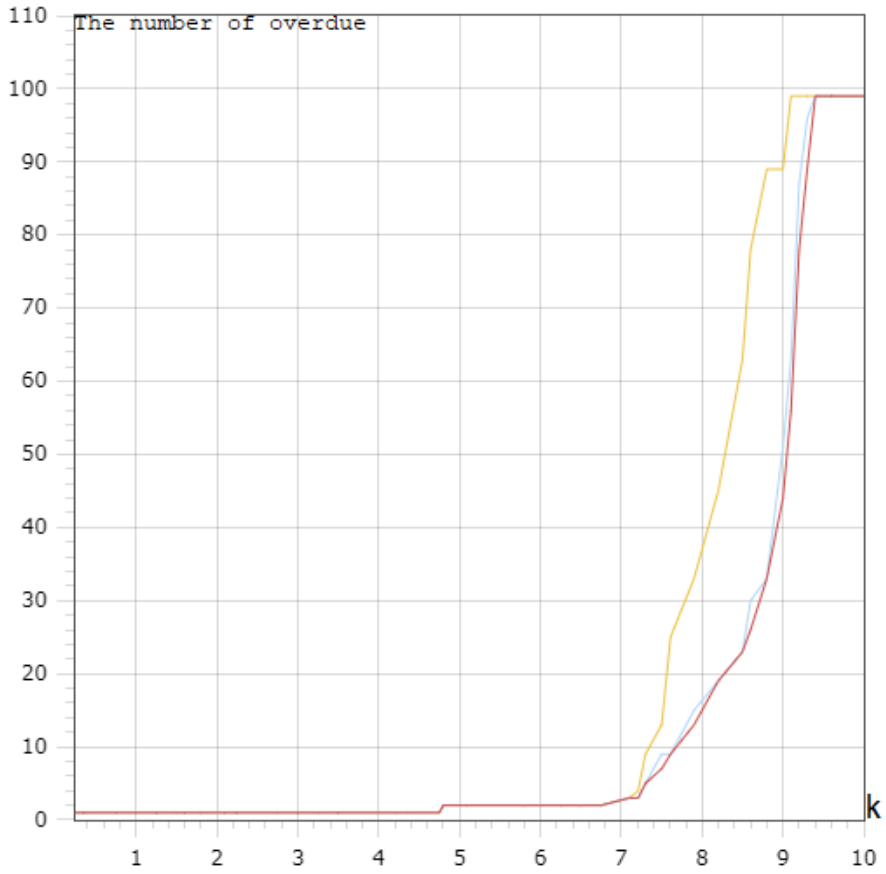
```
      let prevSymbol = prevFlag.symbol;
      for (let j = 0; j < tasksArr.length; j++) {
        const newFlag = flags.find((flag) => flag.name === historyArray[j][i]);
        if (newFlag && newFlag.rank < prevFlag.rank) {
          prevFlag = newFlag;
          prevSymbol = colorize(j + 2)(newFlag.symbol);
        }
      }
      result += prevSymbol;
    }
    result += '\n';
    return (
      result +
      '"<" - added task to queue; "-" - used resource; ">" - completed task; "*" -
missed deadline.'
    );
  };

  addTask = (taskIndex, work) => {
    const { iterate, tasksArr } = this;
    if (taskIndex < 0 || taskIndex > tasksArr.length - 1) return this;
    const ts = this.tasksArr[taskIndex].tStart;
    this.tasksArr[taskIndex].tStart = 1;
    iterate();
    this.tasksArr[taskIndex].tStart = ts;
    work();
    this.counter1++; //
    return this;
  };
}

module.exports = { SchedulingAlgorithm, Task };
```

Результати роботи програми

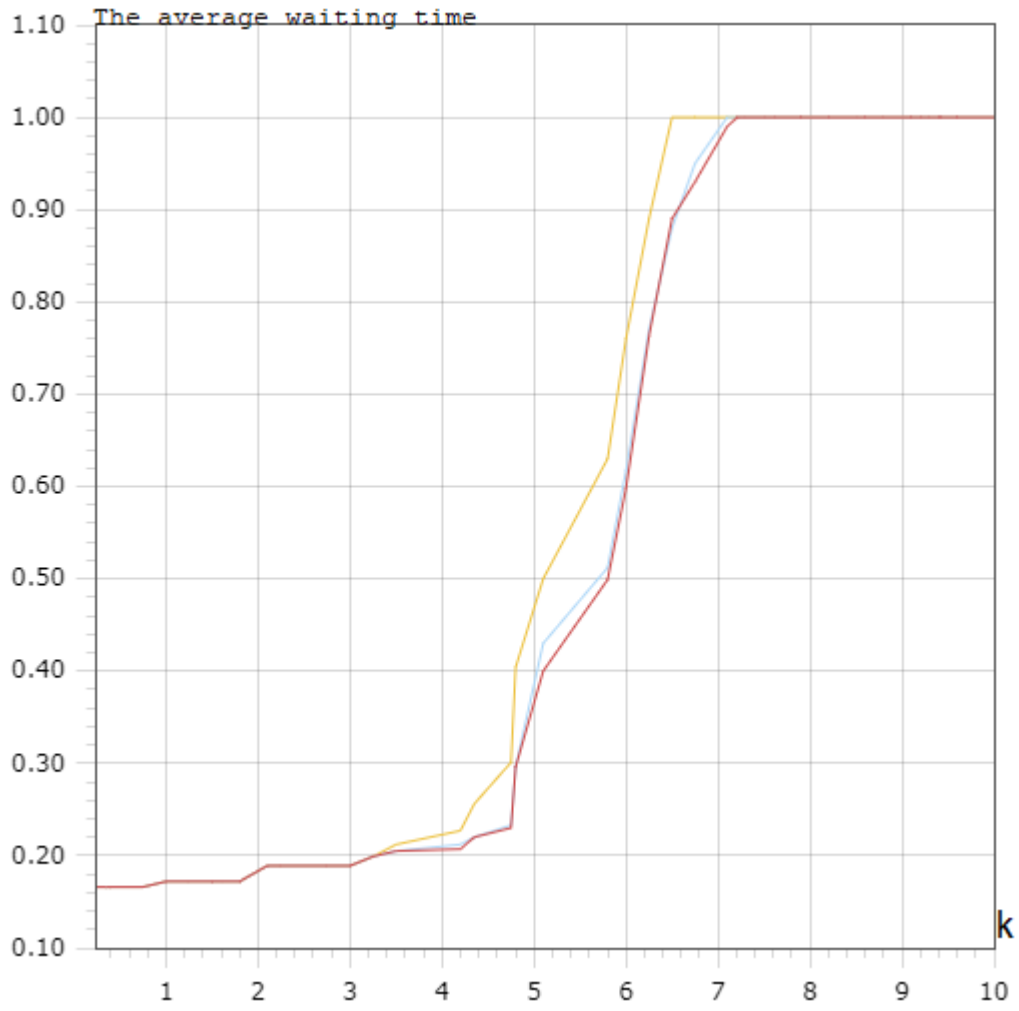
Графік залежності середнього часу очікування від інтенсивності вхідного потоку



Графік відмов вхідних заявок залежно від інтенсивності вхідного потоку заявок



Графік залежності простою ресурсу від інтенсивності вхідного потоку заявок



Висновки

Ми – змоделювали роботу планувальника задач у системі реального часу Визначили середній час очікування від інтенсивності вхідного потоку, залежність простою ресурсу від інтенсивності вхідного потоку заявок, кількість відмов вхідних заявок залежно від інтенсивності вхідного потоку заявок. Дані знання знадобляться нам у нашій подальшій кар’єрі.