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»

Студента 3 курсу групи ІП-84
спеціальності
121 «Інженерія програмного
забезпечення»

Кришталь Д.В.
(прізвище та ініціали)

Керівник доцент Волокіта А.Н.

Київ – 2021 рік

Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"

Факультет (інститут) Інформатики та обчислювальної техніки
(повна назва)

Кафедра Обчислювальної техніки
(повна назва)

Освітньо-кваліфікаційний рівень Бакалавр

Напрям підготовки 121 «Інженерія програмного забезпечення»
(шифр і назва)

З А В Д А Н Н Я
НА РОЗРАХУНКОВО-ГРАФІЧНУ РОБОТУ

Кришталь Дмитру Вікторовичу
(прізвище, ім'я, по батькові)

Тема роботи «Дослідження роботи планувальників роботи систем реального часу»

керівник роботи Волокіта Артем Миколайович к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

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

ЗМІСТ

Основні теоретичні відомості

Планувальник задач (англ. scheduler) – компонент, що забезпечнує виконання завчасно підготовлених завданя у відведений час. Він є ключовим елементов у багатозадачності у системах загального призначення та системах реального часу.

Планувальник запускає виконання різних задач при різних крітеріях, таких як:

- наступ певного часу;
- переходу системи у відповідних стан.

Основною метою планувальника задач є правильне розподілення задач для виконання на певному проміжку часу, тобто створення максимального наванатження на доступні ресурси відповідної системи. Для забезпечення цього система має оператися на наступне:

- 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 ниті

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

Система масового обслуговування (СМО) — система, яка виконує обслуговування вимог (заявок), що надходять до неї. Обслуговування вимог у СМО проводиться обслуговуючими приладами. Класична СМО містить від одного до нескінченного числа приладів. В залежності від наявності можливості очікування вхідними вимогами початку обслуговування СМО (наявності черг) поділяються на:

1) системи з втратами, в яких вимоги, що не знайшли в момент надходження жодного вільного приладу, втрачаються;

2) системи з очікуванням, в яких є накопичувач нескінченної ємності для буферизації надійшли вимог, при цьому очікують вимоги утворюють чергу;

3) системи з накопичувачем кінцевої ємності (чеканням і обмеженнями), в яких довжина черги не може перевищувати ємності накопичувача; при цьому вимога, що надходить в переповнену СМО (відсутні вільні місця для очікування), втрачається.

Основні поняття СМО:

- Вимога (заявка) — запит на обслуговування.
- Вхідний потік вимог — сукупність вимог, що надходять у СМО.
- Час обслуговування - період часу, протягом якого обслуговується вимогу.

Вимоги до системи

Вхідні задачі

Вхідними заявками є обчислення, які проводилися в лабораторних роботах 1-3, а саме обчислення математичного очікування, дисперсії, автокореляції, перетворення Фур'є.

Вхідні заявки характеризуються наступними параметрами:

1) час приходу в систему – Tp – потік заявок є потоком Пуассона або потоком Ерланга k-го порядку (інтенсивність потоків та їх порядок задаються варіантом);

2) час виконання (обробки) – To; математичним очікуванням часу виконання є середнє значення часу виконання відповідних обчислень в попередніх лабораторних роботах;

3) крайній строк завершення (дедлайн) – Td –задається (випадково?); якщо заявка залишається необробленою в момент часу t = Td, то її обробка припиняється і вона покидає систему.

Потік вхідних задач

Потоком Пуассона є послідовність випадкових подій, середнє значення інтервалів між настанням яких є сталою величиною, що дорівнює $1/\lambda$, де λ – інтенсивність потоку.

Потоком Ерланга k-го порядку називається потік, який отримується з потоку Пуассона шляхом збереження кожної (k+1)-ї події (решта відкидаються). Наприклад, якщо зобразити на часовій осі потік Пуассона, поставивши у відповідність кожній події деяку точку, і відкинути з потоку кожну другу подію (точку на осі), то отримаємо потік Ерланга 2-го порядку. Залишивши лише кожну

третю точку і відкинувши дві проміжні, отримаємо потік Ерланга 3-го порядку і т.д. Очевидно, що потоком Ерланга 0-го порядку є потік Пуассона.

Пристрій обслуговування

Пристрій обслуговування складається з P незалежних рівноправних обслуговуючих приладів - обчислювальних ресурсів (процесорів). Кожен ресурс обробляє заявки, які йому надає планувальник та може перебувати у двох станах – вільний та зайнятий. Обробка заявок може виконуватися повністю (заявка перебуває на обчислювальному ресурсі доти, доки не обробиться повністю) або поквантово (ресурс обробляє заявку лише протягом певного часу – кванту обробки – і переходить до обробки наступної заявки).

Пріоритети заявок

Заявки можуть мати пріоритети – явно задані, або обчислені системою (в залежності від алгоритму обслуговування або реалізації це може бути час обслуговування (обчислення), час до дедлайну і т.д.). Заявки в чергах сортуються за пріоритетом. Є два види обробки пріоритетів заявок:

1) без витіснення – якщо в чергу до ресурсу потрапляє заявка з більшим пріоритетом, ніж та, що в даний момент часу обробляється ним, то вона чекає завершення обробки ресурсом його задачі.

2) з витісненням – якщо в чергу до ресурсу потрапляє заявка з більшим пріоритетом, ніж та, що в даний момент часу обробляється ним, то вона витісняє її з обробки; витіснена задача стає в чергу.

Дисципліна обслуговування

Вибір заявки з черги на обслуговування здійснюється за допомогою так званої дисципліни обслуговування. Їх прикладами є FIFO (прийшов першим - обслуговується першим), LIFO (прийшов останнім - обслуговується першим), RANDOM (випадковий вибір). У системах з очікуванням накопичувач в загальному випадку може мати складну структуру.

Дисципліна FIFO

Алгоритм планування First in, first out, також відомий як First come, first served, (перший прийшов, перший вийшов) – один з найпростіших алгоритмів планування. Він просто виконує завдання у тому порядку, якому вони поступають.

Дисципліна EDF

Алгоритм планування Earliest Deadline First (по найближчому строку завершення) використовується для встановлення черги заявок в операційних системах реального часу.

При наставанні події планування (завершився квант часу, прибула нова заявка, завершилася обробка заявки, заявка прострочена) відбувається пошук

найближчої до крайнього часу виконання (дедлайну) заявки і призначення її виконання на перший вільний ресурс або на той, який звільниться найшвидше.

Дисципліна RM

Алгоритм планування Rate-monotonic – алгоритм планування, що відноситься до статичної категорії планування пріорітетів систем реального часу. Він є доволі примітивним. Пріорітетність виконання задачі визначається часом, що є необхідним для її вирішення. Першою виконується задача, що має найкоротший час виконання.

Розробка програми

Мова виконання: JavaScript.

Алгоритми, написані у минулих лабораторних роботих, були винисені у окремий файл signalGenerator.js.

Усі алгоритми планування(FIFO, EDF, RM) та функція для отримання середнього часу очікування написані у файлі algorithms.js. Їх можна переглянути у доатку.

const fifo = (queue) => {…}
const edf = (queue) => {…}
const rm = (queue) => {…}

Усі фукнції отримують чергу як значення агрументу. Виконані задачі переміщаються у масив finishedTasks, а задачі, що не вспіли виконатись у відведених для них час переміщаються у масив failedTasks.

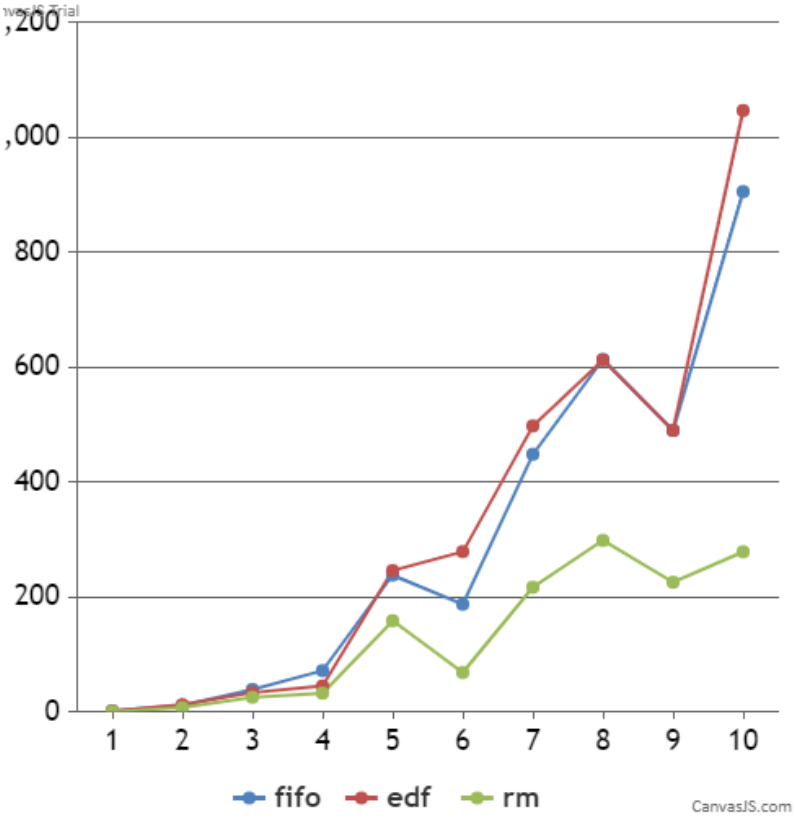
Для виконання планування спочатку необхыдно створити черги. Вони генеруються у функції generateQueue.

const generateQueue = (numberOfTasks, intensity) => {…}.

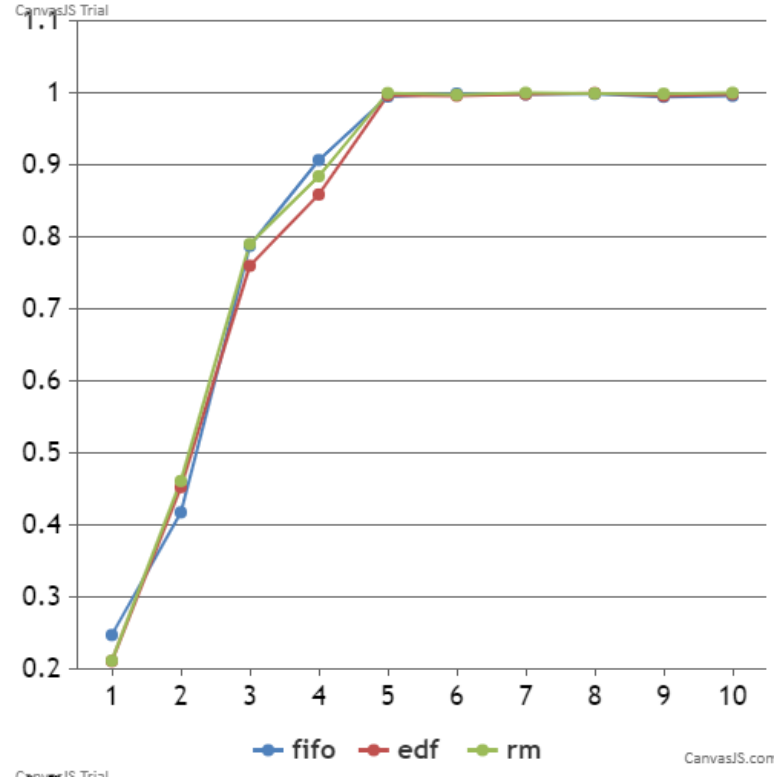
Для побудови графіків було створено drawChart.html.

Результати виконання

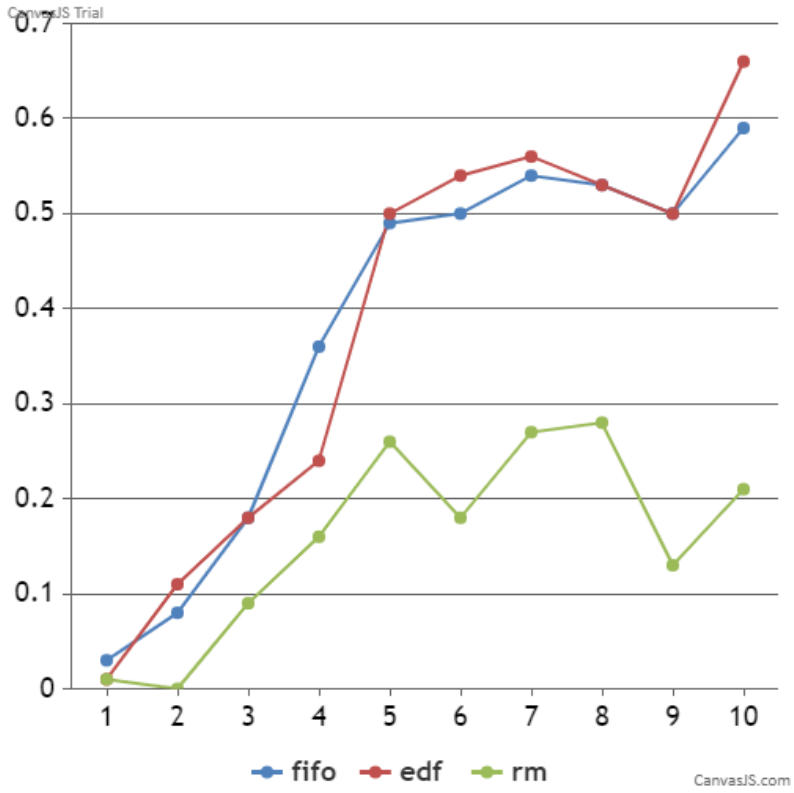
Графік залежності середнього часу очікування від інтенсивності.



Графік залежності проценту простою від інтенсивності.



Графік залежності кількості відмов від інтенсивності.



Додаток

```
let finishedTasks = [];
let failedTasks = [];
let idleTime = 0;
let allTime = 0;

const task1 = (t, arrT, d, aET) => {
    return {
        task: t,
        deadline: d,
        arrivalTime: arrT,
        startTime: 0,
        endtime: 0,
        avarageExecutionTime: aET
    }
}
const fifo = (queue) => {
    finishedTasks = []
    failedTasks = []
    idleTime = 0
    allTime = 0
    let timeWhileExecuting = 0
    const timeStart = Date.now()
    while (queue.length != 0) {
        const timeRan = Date.now() - timeStart
        if(timeRan >= queue[0].arrivalTime){
```

```
            const timeNow = Date.now()
            queue[0].startTime = timeRan
            const execTime = executeTask(queue[0].task)
            queue[0].endtime = queue[0].startTime + (execTime == 0 ? 1: execTime)
            timeWhileExecuting += (Date.now() - timeNow)
            if(queue[0].deadline < queue[0].endtime) {
                console.log("fifo deadline exceeded")
                failedTasks.push(queue.shift())
                continue
            }

        } else {
            continue
        }
        finishedTasks.push(queue.shift())
    }
    allTime = Date.now() - timeStart
    idleTime = allTime - timeWhileExecuting
}

const edf = (queue) => {
    finishedTasks = []
    failedTasks = []
    idleTime = 0
    allTime = 0
    let timeWhileExecuting = 0

    queue.sort((a, b) => {a.deadline - b.deadline})

    const timeStart = Date.now()
    while (queue.length != 0) {
        allTime++;
        const timeRan = Date.now() - timeStart
        let currentIndex = 0
        const currentTask = queue.find(element => {
            currentIndex++
            return element.arrivalTime <= timeRan
        })
        if(currentTask != undefined){
            const timeNow = Date.now()
            currentTask.startTime = timeRan
            const execTime = executeTask(currentTask.task)
            currentTask.endtime = currentTask.startTime + (execTime == 0 ? 1: execTime)
            timeWhileExecuting += (Date.now() - timeNow)
            if(currentTask.deadline < currentTask.endtime) {
                console.log("edf deadline exceeded")
                failedTasks.push(currentTask)
                queue.splice(currentIndex - 1, 1)
                continue
```

```
            }
            finishedTasks.push(currentTask)
            queue.splice(currentIndex - 1, 1)
        } else {
            continue
        }
    }
    allTime = Date.now() - timeStart
    idleTime = allTime - timeWhileExecuting
}

const rm = (queue) => {
    finishedTasks = []
    failedTasks = []
    idleTime = 0
    allTime = 0
    let timeWhileExecuting = 0

    queue.sort((a, b) => a.avarageExecutionTime - b.avarageExecutionTime)
    const timeStart = Date.now()
    while (queue.length != 0) {
        allTime++;
        const timeRan = Date.now() - timeStart
        let currentIndex = 0
        const currentTask = queue.find(element => {
            currentIndex++
            return element.arrivalTime <= timeRan
        })
        if(currentTask != undefined){
            const timeNow = Date.now()
            currentTask.startTime = timeRan
            const execTime = executeTask(currentTask.task)
            currentTask.endtime = currentTask.startTime + (execTime == 0 ? 1: execTime)
            timeWhileExecuting += (Date.now() - timeNow)
            if(currentTask.deadline < currentTask.endtime) {
                console.log("rm deadline exceeded")
                failedTasks.push(currentTask)
                queue.splice(currentIndex - 1, 1)
                continue
            }
            finishedTasks.push(currentTask)
            queue.splice(currentIndex - 1, 1)
        } else {
            idleTime++
            continue
        }
    }
    allTime = Date.now() - timeStart
    idleTime = allTime - timeWhileExecuting
```

```
}

const calculateAvarageWaitingTime = () => {
    let sum = 0;
    for (task of finishedTasks) {
        sum += (task.startTime - task.arrivalTime)
    }
    for(task of failedTasks) {
        sum += (task.startTime - task.arrivalTime)
    }

    return sum/(finishedTasks.length + failedTasks.length)
}

const executeTask = (task) => {

    const signal = getSignal()
    let executionTime;
    switch (task) {
        case "discreteFourier":
            executionTime = discreteFourier(signal)
            break
        case "fastFourier":
            executionTime = fastFourier(signal)
            break
        case "getCorrelation":
            executionTime = getCorrelation(signal)
            break
        case "getMean":
            executionTime = getMean(signal)
            break
        case "getVariance":
            executionTime = getVariance(signal)
            break
    }

    return executionTime
}
```