НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

Звіти до комп’ютерних практикумів з кредитного модуля

“Parallel computing”

**Виконав**

**Студенти групи ІТ-02**

**Терешкович М.О.**

**Перевірив:**

Київ – 2023

**Комп‘ютерний практикум No 5**

**Застосування високорівневих засобів паралельного програмування для побудови алгоритмів імітації та дослідження їх ефективності**

ЗАВДАННЯ

Зображення, що містить текст

Автоматично згенерований опис

**ВИКОНАННЯ**

# Основні класи і методи

Почнемо з того що таке багатоканальна СМО та для чого вона потрібна (надалі СМО або БКСМО):

це система, яка використовується для оптимізації та керування процесом обслуговування клієнтів або користувачів у ситуаціях, коли наявні ресурси недостатні для негайного задоволення потреб всіх користувачів.

У ній присутня можливість обслуговування декількох клієнтів одночасно в окремих каналах. Ці канали можуть бути фізичними (наприклад, окремі каси в магазинах) або віртуальними (наприклад, онлайн каса в кінотеатрах, оператори підтримки).

БКСМО дозволяє знизити час очікування клієнтів, розподіляючи навантаження між каналами обслуговування. Також вона дозволяє оптимізувати використання ресурсів і підвищити продуктивність обслуговування.

Для початку в додаткових вказівках до лабораторної було вказано, що програма має мати елементи управління Producer/Consumer з 3 лр, один з них буде додавати в чергу, а інший забирати. Тому візьмемо їх та трохи модифікуємо для коректнішої роботи.

## Class Producer:

A screen shot of a computer program

Description automatically generated with low confidenceКлас Producer додає в чергу та має конструктор, який приймає параметр типу SystemService і присвоює його змінній екземпляра SystemService (про цей клас буде далі).

Далі йде перевизначення методу run(), який є точкою входу для виконання потоку. У середині методу run() виконуються наступні кроки:

1. Створюється об'єкт Random з іменем random для генерації випадкових чисел.
2. Ініціалізуються дві змінні, startTime і elapsedTime, для відстеження часу.
3. Цикл while виконується до тих пір, поки значення elapsedTime не стане меншим за 10 000 мілісекунд (10 секунд).
4. Усередині циклу рядок systemService.push(random.nextInt(100)) викликає метод push() на об'єкті systemService. Він передає випадково згенероване ціле число від 0 до 99 як аргумент методу push().Метод push(), відповідає за додавання елементів до черги всередині SystemService.
5. Після додавання елемента потік переходить у режим сну на випадкову кількість мілісекунд (до 15 мілісекунд) з використанням Thread.sleep(random.nextInt(15)). Це вводить затримку між послідовними додаваннями.
6. Значення elapsedTime оновлюється шляхом віднімання startTime від поточного часу за допомогою System.currentTimeMillis() - startTime.
7. Як тільки час, що минув, досягає або перевищує 10 000 мілісекунд, цикл завершується, а змінна systemService.isQueueOpen встановлюється у false. Ця змінна вказує на те, що, використовується черга і що вона більше не відкрита для подальших операцій.

Таким чином, клас Producer представляє собою потік, який багаторазово додає випадкові цілі числа (0-99) до черги об'єкта SystemService. Він продовжує робити це протягом максимум 10 секунд, з випадковою затримкою між кожним поштовхом. Після 10 секунд він закриває чергу, встановлюючи systemService.isQueueOpen у false.

## Сlass Consumer:

Клас Consumer забирає з черги, та як і Producer має конструктор, який приймає параметр типу SystemService і присвоює його змінній екземпляра SystemService.

A screen shot of a computer program

Description automatically generated with low confidence

Далі також йде перевизначення методу run():

1. Створюється об'єкт Random з іменем random для генерації випадкових чисел.
2. Потік входить у цикл while, який продовжує виконуватися доти, доки systemService.isQueueOpen має значення true. Це означає, що systemService підтримує булеву змінну isQueueOpen для контролю наявності елементів у черзі.
3. Усередині циклу викликається метод systemService.pop() для отримання елемента з черги, яку обслуговує systemService. Потім, він видаляє елемент з черги і повертає його.
4. Після отримання елемента потік переходить у сплячий режим на випадкову кількість мілісекунд (до 100 мілісекунд) за допомогою Thread.sleep(random.nextInt(100)). Це призводить до затримки між обробкою кожного елемента.
5. Нарешті, викликається метод systemService.incrementApprovedCount(). Цей метод збільшує лічильник, щоб показати, що елемент було успішно оброблено.

Підсумовуючи, клас Consumer представляє потік, який безперервно споживає елементи з черги або структури даних, що підтримується об'єктом SystemService. Він робить це до тих пір, поки прапор isQueueOpen встановлений у true. Між кожним споживанням потік перебуває у стані сну довільної тривалості, також йде відстеження кількості спожитих елементів за допомогою методу incrementApprovedCount().

## Class Analyst:

Клас Analyst має конструктор, який приймає параметр типу SystemService і присвоює його змінній екземпляра systemService. Він також ініціалізує змінні sumQueuesLengths(довжина черги) та ітерації нулем. Потім йде перевизначення методу run();

A screen shot of a computer program

Description automatically generated with low confidence

Усередині методу run() виконуються наступні кроки:

1. Цикл while продовжує виконуватися до тих пір, поки systemService.isQueueOpen є істинним.
2. Усередині циклу потік засинає на 100 мілісекунд за допомогою Thread.sleep(100). Це створює затримку між кожною ітерацією циклу.
3. Метод SystemService.getCurrentQueueLength() викликається для отримання поточної довжини черги.
4. Поточна довжина черги додається до змінної sumQueuesLengths.
5. Змінна ітерації інкрементується для відстеження кількості ітерацій.

Метод getAverageQueueLength() обчислює і повертає середню довжину черги. Він ділить sumQueuesLengths на ітерацію, використовуючи приведення типу (double)iteration, щоб забезпечити ділення з плаваючою комою і отримати більш точне середнє значення.

Як результат, клас Analyst являє собою потік, який безперервно обчислює середню довжину черги об'єктів SystemService, поки черга відкрита (systemService.isQueueOpen має значення true). Періодично він засинає на 100 мілісекунд, отримує поточну довжину черги за допомогою systemService.getCurrentQueueLength() і накопичує довжини, щоб пізніше обчислити середнє значення. Метод getAverageQueueLength() повертає середню довжину черги після завершення роботи потоку.

## Class Watcher:

A screen shot of a computer program

Description automatically generated with low confidence

Клас Watcher має конструктор, який приймає параметр типу SystemService і присвоює його змінній екземпляра systemService.

Метод run():

1. Цикл while продовжує виконуватися до тих пір, поки systemService.isQueueOpen є істинним.
2. Усередині циклу потік засинає на 100 мілісекунд за допомогою Thread.sleep(100). Це створює затримку між кожною ітерацією циклу.
3. Метод SystemService.getCurrentQueueLength() викликається для отримання поточної довжини черги.
4. Поточна довжина черги додається до змінної totalCount.
5. Викликається метод systemService.calculateRejectedPercentage() для обчислення ймовірності відхилення.
6. Розмір черги та ймовірність відмов виводиться на екран за допомогою методу System.out.println().
7. Після виходу з циклу загальний підрахунок виводиться за допомогою System.out.println("Total count: " + totalCount).

Таким чином, клас Watcher представляє собою потік, який безперервно відстежує розмір черги та ймовірність відхилення об'єкта SystemService, поки черга відкрита (systemService.isQueueOpen дорівнює true). Періодично він засинає на 100 мілісекунд, отримує поточну довжину черги за допомогою systemService.getCurrentQueueLength(), обчислює ймовірність відхилення за допомогою systemService.calculateRejectedPercentage() і виводить розмір черги та ймовірність відхилення. Після завершення роботи потік виводить загальну кількість, яка є сумою всіх довжин черг, що спостерігалися під час моніторингу.

## Class SystemService:

A screenshot of a computer program

Description automatically generated with medium confidenceЦе основний код класу SystemService, який інкапсулює функціональність системної служби.

Клас SystemService має наступні змінні:

* QUEUE\_SIZE - максимальний розмір черги.
* rejectedCount - відстежує кількість відхилених елементів.
* approvedCount - відстежує кількість схвалених елементів.
* queue - це реалізація Queue, зокрема ArrayDeque, що використовується для зберігання елементів.
* isQueueOpen - вказує, чи відкрита черга чи ні.

Також присутній конструктор, який приймає параметр QUEUE\_SIZE та ініціалізує змінні екземпляру. Параметри approvedCount і rejectedCount встановлюються в нуль, isQueueOpen встановлюється в true, і створюється новий екземпляр ArrayDeque для ініціалізації черги.

A screen shot of a computer code

Description automatically generated with low confidence

Метод push() відповідає за додавання елементу до черги:

1. Перевіряє, чи розмір черги (queue.size()) більший або дорівнює максимальному розміру черги (QUEUE\_SIZE). Якщо так, він збільшує rejectedCount і повертається, вказуючи, що елемент було відхилено.
2. Якщо черга не заповнена, він додає елемент до черги за допомогою queue.add(item).
3. Метод викликає notifyAll(), щоб розбудити всі потоки очікування, які чекали на появу елементів у черзі.

A screen shot of a computer code

Description automatically generated with low confidence

Метод pop() відповідає за отримання елементу з черги:

1. Він входить у цикл while і продовжує ітерацію до тих пір, поки черга не стане порожньою (queue.size() == 0).
2. Усередині циклу потік чекає за допомогою wait(), поки інший потік не викличе notify() або notifyAll(), вказуючи на наявність елементів у черзі.
3. Як тільки елемент стає доступним у черзі, цикл завершується.
4. Метод отримує і видаляє голову черги за допомогою queue.poll() і повертає значення у викликаючий код.

A picture containing text, screenshot, font, line

Description automatically generated

Метод incrementApprovedCount() відповідає за збільшення змінної approvedCount на одиницю.

A picture containing text, screenshot, font

Description automatically generated

Метод calculateRejectedPercentage() обчислює і повертає відхилений відсоток як десяткове значення. Він ділить rejectedCount на суму rejectedCount і approvedCount, виконуючи приведення типу до типу double для забезпечення ділення з плаваючою комою.

A screenshot of a computer screen

Description automatically generated with low confidence

Метод totalCount() обчислює і повертає загальну кількість оброблених елементів шляхом підсумовування змінних rejectedCount і approvedCount.

A picture containing text, font, screenshot, line

Description automatically generated

Метод getCurrentQueueLength() повертає поточну довжину (кількість елементів) у черзі.

Можемо зробити висновок, що клас SystemService представляє системний сервіс, який керує чергою елементів. Він надає методи для додавання елементів до черги (push()), отримання елементів з черги (pop()), збільшення кількості схвалених елементів (incrementApprovedCount()), обчислення відсотка відхилених елементів (calculateRejectedPercentage()), отримання загальної кількості оброблених елементів (totalCount()) та отримання поточної довжини черги (getCurrentQueueLength()). Методи синхронізовано для забезпечення безпеки потоків при доступі до спільних даних та їх модифікації.

## Class SystemInitializer:

A screen shot of a computer program

Description automatically generated with low confidenceЦей клас SystemInitializer реалізує інтерфейс Callable<double[]> та відповідає за ініціалізацію та запуск системи з декількома компонентами, такими як Producer, Consumer, Watcher, Analyst.

В програмі присутні наступні змінні:

* QUEUE\_SIZE - максимальний розмір черги.
* CHANNELS - кількість каналів споживачів.
* hasWatcher - вказує, чи є в системі компонент-спостерігач.

A picture containing text, screenshot, software

Description automatically generated

SystemInitializer реалізує метод call(), визначений інтерфейсом Callable, і повертає масив типу double.

1. Створює ExecutorService з використанням Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors()). Цей виконавець буде керувати виконанням декількох потоків.
2. Він створює об'єкт SystemService з наданим розміром QUEUE\_SIZE.
3. Створює об'єкт Analyst з об'єктом SystemService.
4. Він запускає вказану кількість потоків Consumer, кожен з яких працює з SystemService.
5. Якщо hasWatcher має значення true, запускає потік Watcher, що працює на SystemService для аналізу роботи.
6. Запускає потік Producer.
7. Запускає потік Analyst.
8. Завершує роботу виконавця, вказуючи на те, що більше не буде надіслано жодного завдання.
9. Виводить повідомлення "SMO is running", щоб показати, що система працює.
10. Очікує завершення роботи виконавця за допомогою методу executor.awaitTermination(30, TimeUnit.SECONDS).
11. Він повертає масив, що містить обчислений відсоток відмов, середню довжину черги від аналітика і загальну кількість оброблених елементів.

Таким чином, клас SystemInitializer являє собою ініціалізатор системи, який налаштовує і запускає систему з декількома компонентами, такими як Producer, Consumer, Watcher, Analyst. Він створює необхідні потоки, призначає їм екземпляри SystemService, керує їх виконанням за допомогою виконавця і повертає масив, що містить відповідні метрики від системи.

## Extra methods:

Також в програмі присутні додатковий метод для виводу відсоток відмов і середній розмір черги на основі наданих вхідних параметрів.

A picture containing screenshot, text, font

Description automatically generated

Метод printStatistic отримує два параметри: failureRate і averageNumberInQueue. Це приватний і статичний метод, що означає, що до нього можна звертатися тільки в межах одного класу і він не вимагає виклику екземпляра класу.

Він виводить відсоток відмов шляхом множення failureRate на 100, округлення до двох десяткових знаків і додавання знаку відсотка ("%"). Виводить середній розмір черги шляхом округлення averageNumberInQueue до двох десяткових знаків. Виведення здійснюється за допомогою System.out.println().

# Завдання 1

Отже в першому завданні треба реалізувати БКСМО з обмеженою чергою.

A picture containing text, screenshot, font

Description automatically generated

У коді визначено дві константи: CHANNELS та QUEUE\_SIZE. Ці константи визначають кількість каналів та максимальний розмір черги відповідно.

Головний метод служить точкою входу в програму. Він викликає метод task1.

A screen shot of a computer code

Description automatically generated with low confidenceМетод task1 представляє собою завдання, яке ініціалізує та запускає систему. Він створює новий об'єкт SystemInitializer з параметрами: hasWatcher, встановленим у false, CHANNELS і QUEUE\_SIZE. Він викликає метод виклику об'єкта SystemInitializer, який налаштовує і запускає систему, повертаючи масив статистики. Повернута статистика зберігається у змінній results. Викликає метод printStatistic, передаючи статистику з масиву results.

В результаті роботи програми отримуємо:   
A screen shot of a computer

Description automatically generated with medium confidence

Тепер давайте збільшимо кількість каналів до 5 і подивимося на середню довжину черги та ймовірність відмови.

A screenshot of a computer

Description automatically generated with medium confidence

# Завдання 2

В другому завданні нам потрібно запустити декілька СМО та вивести середнє статистику по системі. Для початку запустимо з 1 каналом і 10 елементами в черзі.

A screenshot of a computer program

Description automatically generated with low confidence

A screen shot of a computer code

Description automatically generated with low confidence

Метод task2 отримує кількість СМО, systemInstancesCount, який представляє кількість екземплярів системи, що мають бути ініціалізовані та запущені паралельно:

1. Спочатку створює службу ExecutorService за допомогою Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors()). Цей виконавець буде керувати виконанням декількох потоків.
2. Він створює список завдань ArrayList з іменами для зберігання завдань ініціалізації системи.
3. Він повторює ітерацію systemInstancesCount декілька разів і додає до списку завдань завдання SystemInitializer з параметром hasWatcher, встановленим у false, CHANNELS і QUEUE\_SIZE.
4. Він викликає executor.invokeAll(tasks), щоб передати всі завдання виконавцю і отримати список об'єктів типу Future<double[]>, що представляють результати виконання завдань.
5. Далі завершує роботу виконавця, вказуючи, що більше не буде передано жодної задачі.
6. Ініціалізує змінні totalAveragesMessages і totalPercentages для відстеження загальної статистики.
7. Ітерація по списку resultList та отримання результатів з кожного об'єкту Future<double[]> за допомогою методу result.get().
8. Накопичує середню довжину черги та відсоток відмов для кожного екземпляра системи у змінні totalAveragesMessages та totalPercentages відповідно.
9. Обчислює середню статистику шляхом ділення накопичених значень на кількість екземплярів системи (resultList.size()).
10. Викликає метод printStatistic, передаючи йому обчислений середній відсоток відмов і середню довжину черги.

Таким чином, метод task2 паралельно виконує декілька завдань ініціалізації системи, отримує статистику від кожного завдання, обчислює сукупну статистику і виводить її за допомогою методу printStatistic. Він використовує службу виконавця для керування паралельним виконанням завдань і збирає результати за допомогою об'єктів типу Future.

І тепер можемо подививтися на результати виконанні:

Спочатку 5 систем 1 канал 10 елементі в кожній:

A screenshot of a computer

Description automatically generated with medium confidence

5 систем 5 каналів 10 елементів в кожній:

A screenshot of a computer program

Description automatically generated with medium confidence

10 систем 5 каналів 10 елементів в кожній:

A screenshot of a computer

Description automatically generated

# Завдання 3

Реалізуємо додатковий потік, що додамо до внутрішнього пулу потоків СМО. Цей потік матиме доступ до критичних значень СМО та буде виводити дані про них на консоль. Для цього потрібно зазначити в що параметр hasWatcher = true.

A picture containing text, screenshot, font

Description automatically generated

A screen shot of a computer code

Description automatically generated with low confidence

Ось що робить метод task3():

Створює новий об'єкт SystemInitializer з параметрами: hasWatcher, встановленим в true, CHANNELS, і QUEUE\_SIZE. Він викликає метод виклику об'єкта SystemInitializer, який налаштовує і запускає систему зі спостерігачем, повертаючи масив статистики. Повернута статистика зберігається у змінній results. Викликає метод printStatistic, передаючи йому статистику з масиву results.

Таким чином, програма налаштовує і запускає систему зі спостерігачем із заданими параметрами, отримує статистику і виводить її на консоль.

Запускаємо з 1 каналом і 10 елементами в черзі.

A screen shot of a computer screen

Description automatically generated with low confidence

Далі запустимо з 5 каналами та 10 елементами в черзі:

A screen shot of a computer screen

Description automatically generated with low confidence

# ВИСНОВОК

В рамках роботи, ми реалізували тестову систему МО за допомогою засобів багатопоточного програмування Java. Також навчилися проводити статистичні дослідження на таких системах, логувати поточні значення під час виконання. Як результат можемо зробити висновок про те що ефективність алгоритму, довжина черги, ймовірність відмови повністю залежить від параметрів. що більше каналів, то менший шанс відмови. що більше часу обробки, то більший шанс відмови. і останнє, що впливає, - інтервал, з яким надходять нові об'єкти в чергу.