МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

ТЕХНОЛОГІЇ РОЗПОДІЛЕНИХ СИСТЕМ ТА ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

Методичні вказівки
до виконання лабораторних робіт
для підготовки здобувачів першого (бакалаврського) рівня вищої освіти
спеціальності 122 «Комп’ютерні науки»

Київ 2024

УДК 681.3.06.(075)
Т38

Укладач: О. Л. Соловей, канд. техн. наук, доцент

Рецензент О. О. Терентьєв, д-р техн. наук, професор

Відповідальна за випуск Т. А. Гончаренко, канд. техн. наук, доцент, завідувач кафедри інформаційних технологій

Затверджено на засіданні кафедри інформаційних технологій, протокол № 7 від 09 лютого 2024 року.

В авторській редакції.

Технології розподілених систем та паралельних обчислень : Т38 методичні вказівки до виконання лабораторних робіт / уклад. О.Л. Соловей. – Київ : КНУБА, 2024. – 48 с.

Містять зміст, порядок оформлення і вказівки до виконання окремих лабораторних робіт.

Призначено для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 122 «Комп’ютерні науки».

Зміст

Загальні положення

Лабораторні роботи є логічним продовженням лекційного курсу з дисципліни «Технології розподілених систем та паралельних обчислень» і є перехідною ланкою від теоретичного курсу до набуття практичних навичок з розробки програм мовою програмування Java з організацією одночасних обчислень в декількох потоках.

Кожна лабораторна робота містить такі види робіт:

- аналіз умови задачі і розробка підходу до її розв’язку;
- покрокову розробку алгоритму розв’язку і його опис;
- обґрунтування алгоритму;
- написання програми, що реалізує цей алгоритм;
- демонстрація правильної роботи програми на обраному наборі тестів;
- складання і захист звіту.

Лабораторна робота №1.

Визначення ефективності способів обробки даних в обчислювальних системах

Мета роботи: закріпити техніку визначення ефективності способів обробки даних в обчислювальних системах.

Теоретичні відомості

Сучасні процесори мають багатоетапні конвеєри команд. Кожен етап (стадія) конвеєра відповідає іншій дії, що виконує процесор. Якщо конвеєрний пристрій є *l*-стадійним і обробка даних на кожній стадії триває один такт, то для виконання n послідовних операцій на цьому пристрої потрібно витратити $l+n-1$ тактів. Якщо ж цей пристрій використовувати у послідовному режимі, то кількість тактів буде рівна *l*n*. Під час використання векторних команд у формулі для тривалості обробки даних на конвеєрному пристрої додається ще один доданок σ – це час (у тактах), необхідний для ініціалізації векторної команди. Тому загальний час рівний $\sigma + l + n - 1$.

Якщо під ефективністю обробки розуміти реальну продуктивність конвеєрного пристрою, рівну відношенню числа виконаних операцій *n* до часу їх виконання *t(n)* , то залежність продуктивності від довжини вхідних векторів визначається наступним співвідношенням:

$$\pi(n) = \frac{n}{t(n)} = \frac{n}{(l+n-1)\tau} = \frac{1}{(1+(l-1)/n)\tau}, \qquad (1.1)$$

де τ – тривалість такту роботи комп'ютера.

Нехай $t_1, t_2, \ldots, t_l$ – тривалості відповідних стадій конвеєра. Тоді перша операція обробки буде тривати $t_1+\ldots+t_l$ тактів, тому загальна кількість тактів, необхідних для виконання n операцій, рівна $t = t_1+\ldots+t_l + (n-1)t_{max} + \sigma$. Для послідовного процесу обробки даних кількість тактів, необхідна для виконання n операцій, визначається за формулою $t = (t_1+\ldots+t_l)\cdot n$. Прискорення *S(n)* під час виконання n операцій, яке

досягається за рахунок використання конвеєрного способу обробки даних, визначається величиною:

$$S(n) = \frac{n(t_1+\ldots+t_l)}{t_1+\ldots+t_l+(n-1)t_{max}+\sigma} \quad (1.2)$$

Граничне прискорення $S = S(n)$ рівне

$$S(n) = \frac{(t_1+\ldots+t_l)}{t_{max}} \quad (1.3)$$

Залежність продуктивності від довжини вхідних векторів визначається наступним співвідношенням:

$$\pi(n) = \frac{n}{(t_1+\ldots+t_l+(n-1)t_{max}+\sigma)\tau} \quad (1.4)$$

Завдання

Для вхідних даних відповідно варіанту необхідно:

1. Обчислити кількість тактів, необхідну для виконання 2000 операцій обробки даних за умови, що пристрій працює: і) у послідовному режимі; іі) у конвеєрному режимі.

2. Підрахувати пікову продуктивність системи для обох режимів функціонування системи.

3. Визначити найменшу кількість операцій, під час виконання яких у конвеєрному режимі досягається прискорення не менше за (відповідно до варіана) від граничного прискорення.

4. Відповісти на контрольні запитання.

Приклад. Обробка даних на конвеєрному пристрої складається із 5 стадій, тривалості яких рівні 3, 5, 2, 6 та 4 такти відповідно. Визначити ефективність обробки даних відповідно 1 – 3, вважаючи, що ініціалізація конвеєра потребує 2-х тактів та тривалість одного такту складає 5 нс:

Розв'язок. Запишемо характеристики конвеєрного пристрою: $l = 5, \sigma = 2, \tau = 5 \cdot 10^{-9} c, t_1 = 3, t_2 = 5, t_3 = 2, t_4 = 6, t_5 = 4.$

Тоді, $t_{max} = max\{t_1, \ldots, t_5\} = 6.$

Знайдемо шукану кількість тактів у випадку n =1000 у послідовному режимі:

$$t_s(n) = (t_1+\ldots+t_l) \cdot n$$

$$t_s(1000) = (3 + 5 + 2 + 6 + 4) \cdot 1000 = 2000$$

б) у конвеєрному режимі:

$$t_c(n) = (t_1 + \ldots + t_l) + (n-1)t_{max} + \sigma;$$

$$t_c(1000) = 20 + 999 \cdot 6 + 2 = 6016.$$

Підрахуємо пікову продуктивність системи для обох режимів:

$$\pi_s = \frac{1}{(t_1 + \ldots + t_l)\tau} = \frac{1}{20 \cdot 5 \cdot 10^{-9}} = 10^7.$$

$$\pi_c = \frac{1}{t_{max}\tau} = \frac{1}{6 \cdot 5 \cdot 10^{-9}} = 10^7.$$

Визначимо шукану кількість операцій n, яка задовольняє умову $S(n) \geq 0{,}9S$:

$$\frac{20n}{20 + 6(n-1) + 2} \geq 0.9\frac{20}{6}$$

$$\frac{n}{6n + 16} \geq \frac{0.9}{6}$$

$$n \geq \frac{114}{6} = 24$$

Відповідь: n = 24

Варіанти завдань

Варіант № 1

Обробка даних на конвеєрному пристрої складається із 6 стадій, тривалості яких рівні 4, 7, 5, 2, 6 та 4 такти відповідно, ініціалізація конвеєра потребує 3-х тактів та тривалість одного такту складає 2 нс. Визначити характеристики 1-2 для виконання 2000 операцій обробки даних. Визначити найменшу кількість операцій, при виконанні яких у конвеєрному режимі досягається прискорення не менше за 95% від граничного прискорення.

Варіант № 2

Обробка даних на конвеєрному пристрої складається із 7 стадій, тривалості яких рівні 3, 4, 2, 5, 2, 6 та 4 такти відповідно, ініціалізація конвеєра потребує 1-го такту та тривалість одного такту складає 4 нс. Визначити характеристики 1-2 для виконання 500 операцій обробки даних. Визначити найменшу кількість операцій, при виконанні яких у конвеєрному

режимі досягається прискорення не менше за 80% від граничного прискорення.

Варіант № 3

Обробка даних на конвеєрному пристрої складається із 5 стадій, тривалості яких рівні 4, 3, 6 та 4 такти відповідно, ініціалізація конвеєра потребує 2-х тактів та тривалість одного такту складає 1 нс. Визначити характеристики 1-2 для виконання 1500 операцій обробки даних. Визначити найменшу кількість операцій, при виконанні яких у конвеєрному режимі досягається прискорення не менше за 98% від граничного прискорення.

Варіант № 4

Обробка даних на конвеєрному пристрої складається із 7 стадій, тривалості яких рівні 4, 2, 6, 5, 9, 6 та 3 такти відповідно, ініціалізація конвеєра потребує 1-го такту та тривалість одного такту складає 4 нс. Визначити характеристики 1-2 для виконання 800 операцій обробки даних. Визначити діапазон для числа операцій, при виконанні яких у конвеєрному режимі досягається прискорення, яке складає від 85% до 95% граничного прискорення.

Варіант № 5

Обробка даних на конвеєрному пристрої складається із 6 стадій, тривалості яких рівні 2, 3, 1, 9 та 3 такти відповідно, ініціалізація конвеєра потребує 2-х тактів та тривалість одного такту складає 2 нс. Визначити характеристики 1-2 для виконання 3000 операцій обробки даних. Визначити діапазон для числа операцій, при виконанні яких у конвеєрному режимі досягається прискорення, не менше за 85% від граничного прискорення.

Варіант № 6

Обробка даних на конвеєрному пристрої складається із 7 стадій, тривалості яких рівні 3, 10, 2, 7, 2, 6 та 3 такти відповідно, ініціалізація конвеєра потребує 4-х тактів та тривалість одного такту складає 1 нс. Визначити характеристики 1-2 для виконання 500 операцій обробки даних.

Визначити діапазон для числа операцій, при виконанні яких у конвеєрному режимі досягається прискорення, яке складає від 85% до 95% граничного прискорення.

Варіант № 7

Обробка даних на конвеєрному пристрої складається із 6 стадій, тривалості яких рівні 6, 7, 5, 10, 6 та 3 такти відповідно, ініціалізація конвеєра потребує 3-х тактів та тривалість одного такту складає 1 нс. Визначити характеристики 1-2 для виконання 3000 операцій обробки даних. Визначити діапазон для числа операцій, при виконанні яких у конвеєрному режимі досягається прискорення, не менше за 85% від граничного прискорення.

Варіант № 8

Обробка даних на конвеєрному пристрої складається із 8 стадій, тривалості яких рівні 2, 3, 5, 6, 2, 6, 4, 2 такти відповідно, ініціалізація конвеєра потребує 3-х тактів та тривалість одного такту складає 0,5 нс. Визначити характеристики 1-2 для виконання 1500 операцій обробки даних. Визначити діапазон для числа операцій, при виконанні яких у конвеєрному режимі досягається прискорення, не менше за 90% від граничного прискорення.

Варіант № 9

Обробка даних на конвеєрному пристрої складається із 5 стадій, тривалості яких рівні 4, 3, 8, 6, 4 такти відповідно, ініціалізація конвеєра потребує 1-го такту з тривалістю 6 нс. Визначити характеристики 1-2 для виконання 700 операцій обробки даних. Визначити діапазон для числа операцій, при виконанні яких у конвеєрному режимі досягається прискорення, не менше за 86% від граничного прискорення.

Варіант № 10

Обробка даних на конвеєрному пристрої складається із 5 стадій, тривалості яких рівні 3, 8, 5,4, 6 та 2 такти, ініціалізація конвеєра потребує 2-х тактів та тривалість одного такту – 2 нс. Визначити характеристики 1-2 для виконання 4000 операцій обробки даних. Визначити діапазон для числа операцій, при виконанні яких у конвеєрному режимі досягається прискорення, не менше за 80% від граничного прискорення.

Контрольні запитання

1. Дайте визначення поняттю «паралелізм».
2. Наведіть класифікацію паралельності за рівнями та наведіть приклади систем для кожного рівня.
3. Який графік має залежність продуктивності конвеєрного пристрою від довжини вхідного набору?
4. Що означає доданок σ у формулі продуктивності?

Лабораторна робота №2.

Характеристики систем функціональних пристроїв

Мета роботи: закріпити техніку визначення характеристик систем функціональних пристроїв.

Теоретичні відомості

Будь-яка обчислювальна система являє собою сукупність деяких функціональних пристроїв (ФП). Для оцінки якості її роботи вводяться характеристики: реальна продуктивність та пікова продуктивність.

Реальна продуктивність системи пристроїв – кількість операцій, які реально виконуються у середньому за одиницю часу. Реальна продуктивність системи пристроїв визначається $r = \sum_{i=1}^{l} \quad p_i\pi_i$, де π_i – пікова продуктивність; p_i завантаженість i-го пристрою. Завантаженість системи є величина

$$p = \sum_{i=1}^{l} \quad \propto_i p_i, \text{ де } \propto_i = \frac{\pi_i}{\sum_{j=1}^{l} \quad \pi_i}, \tag{2.1}$$

Пікова продуктивність – максимальна кількість операцій, які могла би виконати система за одиницю часу у випадку відсутності звя’зків між її ФП. Пікова продуктивність рівна $\frac{1}{t_{max} \cdot \tau}$.

Прискоренням реалізації алгоритму на даній обчислювальній системі (або просто прискоренням) називають $S = \frac{r}{max\pi_i}$. Максимальна продуктивність системи r_{max} виражається формулою:

$$r_{max} = l \cdot min\pi_i, \text{ де } 1 \le i \le l, \tag{2.2}$$

Завдання

Для графу системи функціональних пристроїв (ФП) наведений на рис. 2 і пікових продуктивностей пристроїв системи (відповідно до варіанта) необхідно визначити: 1) завантаженості усіх пристроїв системи; 2) реальну продуктивність кожної системи; 3) завантаженість системи; 4) прискорення системи. Відповісти на контрольні запитання.

Приклад. Граф системи ФП наведений на рис. 2.1. Відомі пікові продуктивності пристроїв системи:

№	0	1	2	3	4	5	6	7	8	9	10	11	12
π	10	5	8	6	7	9	12	8	10	4	6	4	6

Знайти характеристики 1 – 4?

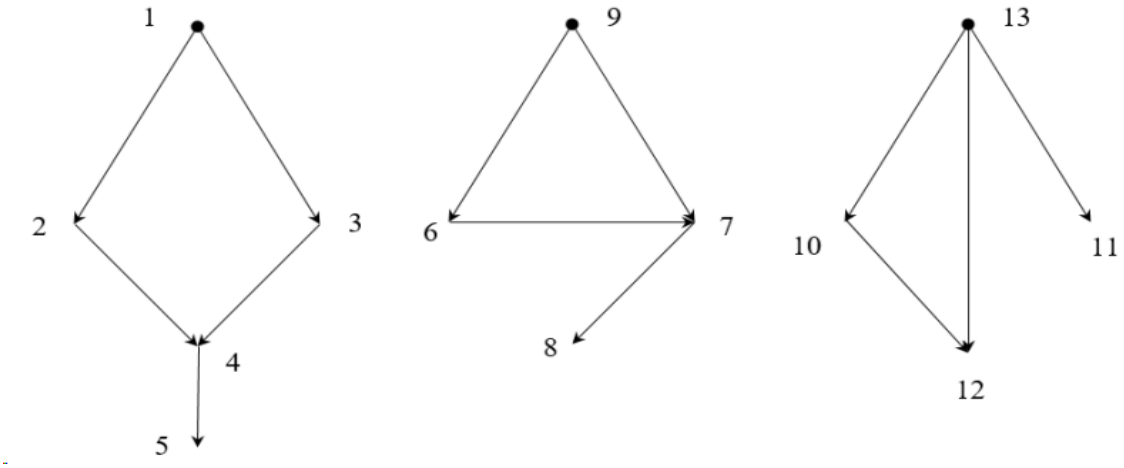

Рис. 2.1. Граф системи ФП

Розв'язок. Як видно з рис. 2.1, система складається з трьох незалежних підсистем. Згідно до 1-го закону Амдала реальна продуктивність кожної із підсистем визначається продуктивністю найменш продуктивного пристрою цієї підсистеми.

Нехай $\underline{r}^{(i)}$ – реальні продуктивності, з якими працюють усі пристрої *i*-системи, *i* =1,2,3. Тобто,

$$r_1 = \ldots = r_5 = \underline{r}^{(1)},$$

$$r_6 = \ldots = r_9 = \underline{r}^{(2)},$$

$$r_{10} = \ldots = r_{13} = \underline{r}^{(3)}.$$

Тоді $\underline{r}^{(1)} = 5, \underline{r}^{(2)} = 8, \underline{r}^{(3)} = 4$

Завантаженості p_i пристроїв першої підсистеми рівні $\underline{r}^{(1)}/\pi_i$, (i=1,…5):

$$p_1 = \frac{5}{10},\ p_2 = 1; p_3 = \frac{5}{8}; p_4 = \frac{5}{6},\ p_5 = \frac{5}{7};$$

Завантаженості p_i пристроїв другої підсистеми рівні $\underline{r}^{(2)}/\pi_i$, (i=6,…9):

$$p_6 = \frac{8}{9},\ p_7 = \frac{2}{3};\ p_8 = 1;\ p_9 = \frac{4}{5},$$

Завантаженості p_i пристроїв третьої підсистеми рівні $\underline{r}^{(3)}/\pi_i$, (i=10,…13):

$$p_{10} = 1,\ p_{11} = \frac{2}{3};\ p_{12} = 1;\ p_{13} = \frac{2}{3},$$

За першим законом Амдала (див. твердження 2.4) $r^{(i)} = l_i \cdot \underline{r}^{(1)}$, де l_i – кількість пристроїв і-ї підсистеми. Тому

$$\underline{r}^{(1)} = 5 \cdot 5 = 25,\ \underline{r}^{(2)} = 4 \cdot 8 = 32,\ \underline{r}^{(3)} = 4 \cdot 4 = 16$$

Отже, реальна продуктивність r = 25+32 + 16=73.

Для знаходження завантаженості системи використаємо формулу $r = p \cdot \pi$, де p – завантаженість, □ – пікова продуктивність системи. Тоді

$$\pi = \pi_1 + \ldots + \pi_{13} =$$

$$= 10 + 5 + 8 + 6 + 7 + 9 + 12 + 8 + 10 + 4 + 6 + 4 + 6 = 95$$

Тому $p = \frac{r}{\pi} = \frac{73}{95} = 0{,}77$

Для знаходження прискорення системи скористаємося формулою:

$$S = r/\pi_i\ ,$$

тоді S=73/12=6,08.

Варіанти завдань

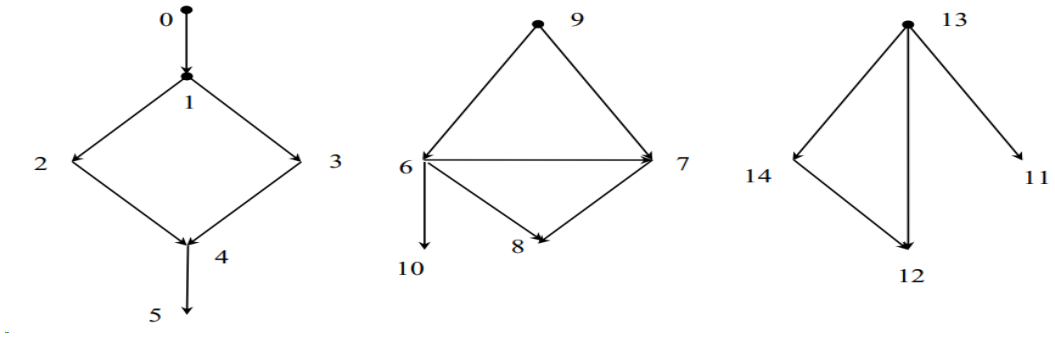

Рис 2.2. Граф системи ФП

Варіанти

Варіант № 1

Пікові продуктивності пристроїв системи.

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
π	5	8	4	6	7	6	9	12	9	15	4	8	10	8	11

Варіант № 2

Пікові продуктивності пристроїв системи.

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
π	7	5	10	8	6	6	9	12	8	5	11	8	7	8	8

Варіант № 3

Пікові продуктивності пристроїв системи.

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
π	15	9	4	8	7	6	9	12	8	12	7	8	10	8	11

Варіант № 4

Пікові продуктивності пристроїв системи.

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
π	9	6	12	7	5	8	3	10	8	7	11	9	7	8	8

Варіант № 5

Пікові продуктивності пристроїв системи.

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
π	9	8	5	8	7	6	9	12	9	15	7	8	10	8	11

Варіант № 6

Пікові продуктивності пристроїв системи.

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
π	4	3	10	8	2	4	9	3	8	9	11	8	6	8	8

Варіант № 7

Пікові продуктивності пристроїв системи.

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
π	7	5	4	7	5	8	7	12	9	5	14	8	10	8	11

Варіант № 8

Пікові продуктивності пристроїв системи.

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
π	7	5	10	8	6	6	9	12	8	5	11	8	7	8	8

Варіант № 9

Пікові продуктивності пристроїв системи.

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
π	8	8	7	5	9	10	9	12	7	14	5	8	12	8	14

Варіант № 10

Пікові продуктивності пристроїв системи.

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
π	5	10	8	6	7	6	9	12	8	5	11	8	7	8	8

Контрольні запитання

1. Дайте визначення терміну «реальна продуктивність системи».
2. За якої умови асимптотична продуктивність буде максимальною?
3. На основі класифікації Флінна назвіть чотири класи архітектури.
4. На основі запропонованої Фенгом класифікації перечисліть чотири класи комп’ютерів.

Лабораторна робота №3.

Граф алгоритму та концепції паралелізму

Мета роботи: закріпити техніку побудови та аналізу паралельних форм обчислювальних алгоритмів.

Теоретичні відомості

Якщо система містить *l* однакових простих універсальних процесорів, то прискорення реалізації алгоритму на цій системі $S_l(n)$ можна обчислити за формулою

$$S_l(n) = \frac{T_1(n)}{T_l(n)}, \qquad (3.1.)$$

де $T_1(n)$ – час, за який можна реалізувати алгоритм на одному процесорі, $T_l(n)$ – час реалізації алгоритму в системі (висота алгоритму), n — розмірність задачі.

Ефективністю $E_l(n)$ реалізації алгоритму у паралельній системі називається відношення прискорення до кількості процесорів системи, тобто

$$E_l(n) = \frac{S_l(n)}{l}, \qquad (3.2)$$

Завдання

Зобразити граф алгоритму паралельного обчислення значення виразу відповідно до варіанта на обчислювальному пристрої з кількістю процесорів відповідно до варіанта. Для кожної паралельної форми обчислити її висоту, ширину, прискорення та ефективність реалізації алгоритму. Відповісти на контрольні запитання.

Приклад. Зобразити граф алгоритму паралельного обчислення значення виразу

$$a_1a_2 + a_2a_3 + a_3a_4 + a_4a_5 + a_5a_6 + a_6a_7 + a_7a_1$$

на обчислювальному пристрої:

1) із одним універсальним процесором;

2) із трьома універсальними процесорами.

Розв'язок. Будемо вважати, що у алгоритмі спочатку обчислюються зліва направо усі 7 добутків, а потім – 6 сум (у такому самому порядку). На рис. 3.1, 3.2 наведено реалізацію алгоритму у послідовній системі, у системі з трьома процесорами відповідно.

Визначимо характеристики реалізації алгоритму при $l = 1$ маємо n=7, $T_1(7) = 13$, $S_1(7) = 1$, $E_1(7) = 1$.

Для $l = 3$ маємо n=7, $T_3(7) = 6$, $S_3(7) = 13/6 \approx 2{,}17$, $E_3(7) = \frac{13}{6 \cdot 3} = 0{,}72$.

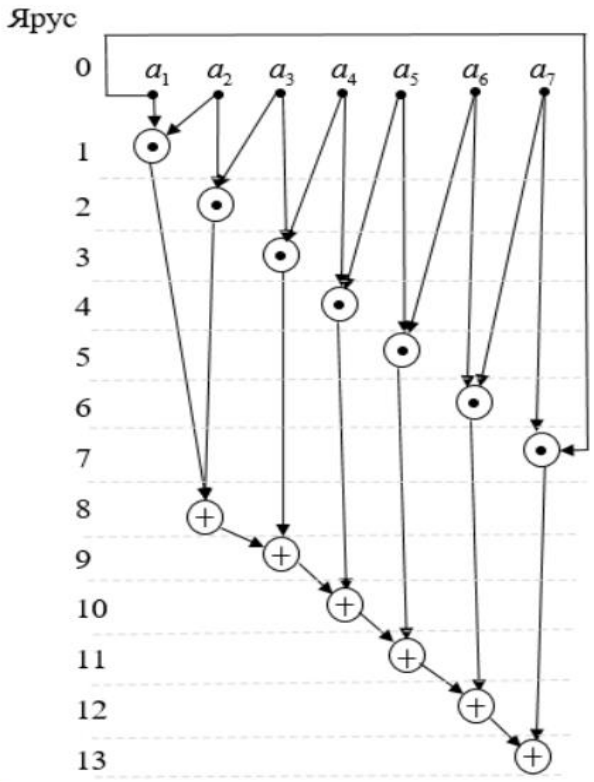

Рис 3.1. Граф алгоритму у послідовній системі

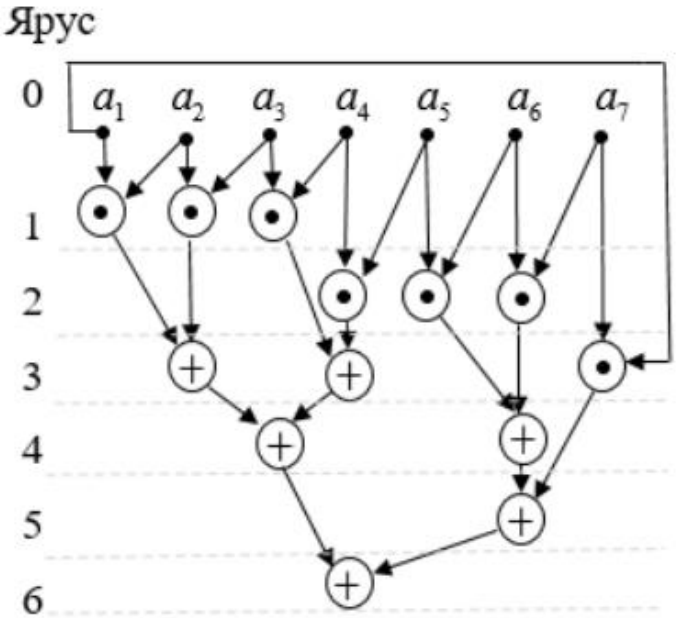

Рис. 3.2. Граф алгоритму у послідовній системі з трьома процесорами

Варіанти завдань

Варіант № 1

Зобразити граф алгоритму паралельного обчислення значення виразу

$$\frac{(a+b+c+d)\frac{e+f}{g+h}}{\frac{\frac{i-j}{k\cdot l}}{m\cdot n+\frac{q}{r}}}, \tag{3.4}$$

Варіант № 2

Зобразити граф алгоритму паралельного обчислення значення виразу

$$\frac{\frac{a}{b}-(c+d)+\frac{e+f}{g\cdot h}}{\frac{i\cdot j}{k\cdot l}(m\cdot n-p+q)}, \tag{3.5}$$

Варіант № 3

Зобразити граф алгоритму паралельного обчислення значення виразу

$$\frac{\left(\frac{a}{b}+\frac{c}{d}\right)\cdot\left(\frac{e+f}{g\cdot h}\right)}{\frac{i\cdot j}{k\cdot l}\left(\frac{m}{n}-\frac{p}{q}\right)}, \tag{3.6}$$

Варіант № 4

Зобразити граф алгоритму паралельного обчислення значення виразу

$$\frac{\left(\frac{a+b}{c\cdot b}+\frac{c}{d}\right)\cdot\frac{i-j}{k-l}}{\frac{e\cdot f}{g-h}\left(\frac{m}{n}-\frac{p}{q}\right)}, \quad (3.7)$$

Варіант № 5

Зобразити граф алгоритму паралельного обчислення значення виразу

$$\frac{\left(\frac{a}{b}+\frac{c}{d}\right)}{(e-f)\cdot\frac{g}{h}}+\frac{\frac{e}{f}\cdot\frac{g}{h}}{\frac{m-n}{p+q}}, \quad (3.8)$$

Варіант № 6

Зобразити граф алгоритму паралельного обчислення значення виразу

$$\left(\frac{a+b}{c+d}-\frac{e}{f}\cdot(g+h)\right)\cdot\left((i\cdot j+k\cdot l)+\left(\frac{m}{n}-(p-q)\right)\right), \quad (3.9)$$

Варіант № 7

Зобразити граф алгоритму паралельного обчислення значення виразу

$$\left(\left(a\cdot b-\frac{c}{d}\right)-(e\cdot f-g\cdot h)\right)+(i\cdot j+k\cdot l+\frac{m\cdot n}{p\cdot q}), \quad (3.10)$$

Варіант № 8

Зобразити граф алгоритму паралельного обчислення значення виразу

$$\frac{a\cdot b}{c+d}\cdot\frac{e\cdot f}{g-h}-\frac{i}{j}\cdot\frac{k}{l}\cdot\left(\frac{m}{n}+\frac{p}{q}\right), \quad (3.11)$$

Варіант № 9

Зобразити граф алгоритму паралельного обчислення значення виразу

$$\frac{a\cdot b}{c-d}-\frac{e\cdot f}{g-h}+\left(\frac{i}{j}+k\cdot l\right)\cdot\left(\frac{m\cdot n}{p\cdot q}\right), \quad (3.12)$$

Варіант № 10

Зобразити граф алгоритму паралельного обчислення значення виразу

$$\left(\frac{a+b}{c\cdot d}-\left(\frac{e}{f}-\frac{g}{h}\right)\right)\cdot\left(\frac{i-j}{k-l}+\frac{m}{n}\cdot\frac{p}{q}\right), \qquad (3.13)$$

Контрольні запитання

1. Що називається висотою паралельної форми?
2. Сформулюйте основні твердження концепції необмеженого паралелізму.
3. Чому в загальному випадку дорівнює висота паралельної форми?
4. Надайте поняття «внутрішній паралелізм».

Лабораторна робота №4.

Розробка програми розв'язку системи рівнянь методом Крамера в паралельних потоках з різними пріоритетами

Мета роботи: вивчити засоби Java для організації одночасних обчислень в декількох потоках виконання з різними пріоритетами.

Теоретичні відомості

Набори операторів, що представляють логічні частини програми, які повинні працювати паралельно один з одним інкапсулюються в об'єкти, звані потоками виконання, а така реалізація паралелізму називається багатопотоковою (multithreading).

Однопотокова програма (single-thread) має одну точку входу (метод main()) і одну точку виходу. Багатопотокова програма (multithread) має початкову точку входу (метод main()) і кілька наступних точок входу і виходу (по кількості потоків), при цьому виконання одного потоку може бути перервано для виконання іншого (рис. 4.1)

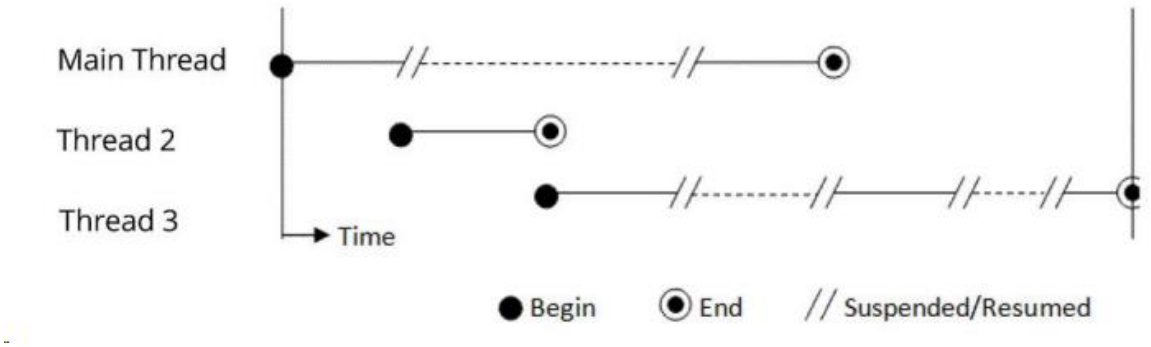

Рис. 4.1. Багатопотокова програма

Потік може бути розподілений для виконання одному виконуючому блоку (процесору), але також може реалізовуватися в багатопроцесорному (багатоядерному) середовищі. Багатозадачні операційні системи зазвичай підтримують багатопоточність.

Існує два способи визначення послідовності операторів, які повинні виконуватись у потоці: 1) написання класу користувача, що реалізовує інтерфейс java.lang.Runnable, при цьому логічна послідовність операторів,

які повинні виконуватись у потоці, зазначається як реалізація абстрактного методу public abstract void run(); 2) написання класу користувача, що успадковує клас java.lang.Thread (останній огортає private Runnable target та реалізує метод run() інтерфейсу Runnable), і перевизначення методу run(), вказавши в ньому логічну послідовність операторів, які повинні виконуватись у потоці.

Таким чином, інтерфейс Runnable – є абстракцією над завданням, що виконується у потоці, і дозволяє розмежити виконання завдання від логіки управління потоками. Окрім того, використання інтерфейсу Runnable дозволяє класу користувача бути спадкоємцем іншого класу. Клас java.lang.Thread визначає ряд методів, які використовуються для управління потоками. До них відносяться статичні методи, які надають інформацію про стан потоку або змінюють стан потоку.

Завдання

Напишіть програму для знаходження коренів системи рівнянь розміром 3x3 методом Крамера. Визначник основної матриці A має обчислюватися в основному потоці Thread-0, визначники матриць A_1, A_2, A_3 в паралельних потоках з більшим пріоритетом, ніж встановлено для основаного потоку. Основний потік має чекати закінчення виконання всіх паралельних потоків і потім обчислювати корені і виводити результати на екран. Реалізуйте метод printInfo, який виводить стан кожного потоку, пріоритет та результат роботи. Відповісти на контрольні запитання.

Приклад. Програма може складатися з наступних класів: CramersRuleThread – успадковує клас java.lang.Thread і перевизнає метод run() та реалізує метод для обчислення визначника. CramersRuleWithPriority – реалізує обчислення в паралельних потоках з визначеними пріоритетам.

```
class CramersRuleThread extends Thread {
    private static final int SIZE = 3;
    private double[][] matrix;
    private double[] column;
    private double determinant;
    public CramersRuleThread(double[][] matrix, double[] column) {
        this.matrix = matrix;
        this.column = column;
        this.determinant = 0.0;
    }
    @Override
    public void run() {
        determinant = calculateDeterminant(matrix, column);
    }
    public double getDeterminant() {
        return determinant;
    }
    private double calculateDeterminant(double[][] matrix, double[] column) {
      // треба додати код
    }
}
public class CramersRuleWithPriority {
    public static void main(String[] args) {
        double[][] coefficients = {
                {2, -1, 3},
                {1, 2, 1},
                {3, 3, 2}
        };
        double[] constants = {8, 5, 10};
```

```
    CramersRuleThread threadX = new CramersRuleThread(replaceColumn(coefficients, constants, 0), constants);
    CramersRuleThread threadY = new CramersRuleThread(replaceColumn(coefficients, constants, 1), constants);
    CramersRuleThread threadZ = new CramersRuleThread(replaceColumn(coefficients, constants, 2), constants);
        threadX.setPriority(Thread.MIN_PRIORITY);
    threadY.setPriority(Thread.NORM_PRIORITY);
    threadZ.setPriority(Thread.MAX_PRIORITY);
    threadX.start();
    threadY.start();
    threadZ.start();
    try {

       threadX.join();
       threadY.join();
       threadZ.join();
       double determinantX = threadX.getDeterminant();
       double determinantY = threadY.getDeterminant();
       double determinantZ = threadZ.getDeterminant();
       System.out.println("Determinant X: " + determinantX);
       System.out.println("Determinant Y: " + determinantY);
       System.out.println("Determinant Z: " + determinantZ);
       double determinantA = // треба додати код;
       double solutionX = // треба додати код;
       double solutionY = // треба додати код;
       double solutionZ = // треба додати код;
```

```
            System.out.println("Solution: x = " + solutionX + ", y = " + solutionY
+ ", z = " + solutionZ);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    private static double calculateDeterminant(double[][] matrix, double[]
column) {
// треба додати код
    }

    private static double[][] replaceColumn(double[][] matrix, double[] column,
int columnIndex) {
        // треба додати код
        return result;
    }
}
```

Варіанти завдань

Програма має розв’язувати систему рівнянь, яку визначає користувач.

Контрольні запитання

1. Що в інформатиці називають паралелізмом?
2. Дайте визначення потокам виконання (threads). У чому їх відмінність від процесів (програм)?
3. Назвіть методи для роботи з потоками, наявні у класі java.lang.Object.
4. Опишіть спосіб створення та запуску потоку із використанням інтерфейсу java.lang.Runnable.
5. Опишіть спосіб створення та запуску потоку із використанням інтерфейсу java.lang.Thread.
6. Опишіть методи класу java.lang.Thread, які використовуються для ідентифікації потоку.
7. Поясніть, що являє собою пріоритет потоків і як він може встановлюватись? Як планувальник потоків враховує їхній пріоритет?

Варіанти завдань

Програма має розв’язувати систему рівнянь, визначену користувачем.

Лабораторна робота №5.

Розробка програми з координацією паралельних потоків та механізми синхронного доступу до критичної секції

Мета роботи: вивчити засоби Java для організації координації паралельних потоків та синхронного доступу до критичної секції.

Теоретичні відомості

У паралельному програмуванні часто доступ для читання і запису до одного ресурсу мають декілька потоків, що може призвести до помилкових ситуацій. Запобігання таких помилок виконується з використанням критичної секції. Критична секція являє собою блок коду, який отримує доступ до загального ресурсу і не може виконуватися більш ніж одним потоком одночасно. Додавання модифікатора synchronized в оголошенні методу позначає його тіло як критичну секцію і забезпечує одночасний доступ до методу тільки для одного потоку.

Таким чином як механізми синхронного доступу до критичної секції Java пропонує: 1) використання ключового слова synchronized для позначки методу або блоку коду як критичної секції; 2) використання реалізацій інтерфейсу Lock і синхронізаторів. Кожен об’єкт в Java має асоційований з ним монітор (високорівневий механізм взаємодії і синхронізації). Монітор є свого роду інструментом управління доступом до об’єкта. Коли виконання коду доходить до оператора synchronized(obj), монітор об’єкта obj блокується, і на час його блокування монопольний доступ до блоку коду має тільки один потік, який і зробив блокування. Після закінчення роботи блоку коду монітор об’єкта obj звільняється і стає доступним для інших потоків.

Синхронізація методів призводить до деякого збільшення часу обчислень порівняно з несинхронізованими методами, але запобігає помилкам. Найбільш ефективно використовувати методи public final void wait() throws InterruptedException, public final native void notify(), public final native void notifyAll() класу Object, щоб призупинити роботу поточного

потоку. Коли потік викликає з об’єкта-монітора синхронізованого блоку коду метод wait(), JVM переводить потік в стан сну і звільняє об’єкт-монітор, що захищає синхронізований блок коду (або синхронізований метод), який потік виконує. Це дозволяє іншим потокам захоплювати звільнений об’єкт-монітор і виконувати блоки синхронізованого коду, захищеного цим об’єктом. Для пробудження переведеного в стан сну потоку необхідно викликати метод notify() або notifyAll() з того ж об’єкта-монітора всередині синхронізованого блоку коду (або синхронізованого методу), що знаходяться під захистом того ж об’єкта-монітора. У разі виклику wait() поза межами синхронізованого блоку (синхронізованого методу) JVM генерує виняток IllegalMonitorStateException.

Завдання

Напишіть програму обчислення добутку 2-х матриць за умови, що кожен рядок матриці A перемножується на стовпець матриці в окремому потоці. В основному потоці вивести результат добутку матриць.

Приклад. Програма може складатися з наступних класів: MatrixComputationThread успадковує клас java.lang.Thread і перевизнає метод run() та реалізує метод критичної секції calculateElement, в якому обчислюється добуток рядка матриці A на стовпець матриці B. Класс MatrixComputationWithSynchronization включає метод main(), в якому створюються потоки, а допомогою виклику методу Executors.newFixedThreadPool() та виконуються обчислення в кожному потоці за допомогою виклику методів MatrixComputationThread(); та executor.execute(thread);

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
class MatrixComputationThread extends Thread {
    private static int[][] matrixA;
    private static int[][] matrixB;
    private static int[][] result;
    private static int numRows;
    private static int numCols;
    private int startRow;
    private int endRow;
    public MatrixComputationThread(int startRow, int endRow) {
        this.startRow = startRow;
        this.endRow = endRow;
    }
    @Override
    public void run() {
        for (int i = startRow; i < endRow; i++) {
            for (int j = 0; j < numCols; j++) {
                result[i][j] = calculateElement(i, j);
            }
        }
    }
    private synchronized int calculateElement(int row, int col) {
        //додайте код для обчислення добутку рядка на стовпчик
    }
    public static void setMatrices(int[][] a, int[][] b) {
        //додайте код для ініціалізації матриць matrixA, та matrixB
    }
    public static int[][] getResult() {
```

```
        //додайте код, який повертає результуючу матрицю
    }
}
public class MatrixComputationWithSynchronization {
    public static void main(String[] args) {
        int[][] matrixA = {
                {1, 2, 3},
                {4, 5, 6},
                {7, 8, 9}
        };
        int[][] matrixB = {
                {9, 8, 7},
                {6, 5, 4},
                {3, 2, 1}
        };
        int numRows = matrixA.length;
        int numCols = matrixB[0].length;
        int[][] result = new int[numRows][numCols];
        MatrixComputationThread.setMatrices(matrixA, matrixB);
        int numThreads = 3;
// додайте код для створення потоків за допомогою виклику методу Executors.newFixedThreadPool()

        for (int i = 0; i < numThreads; i++) {
//додайте свій код для виконання обчислень в кожному потоці за допомогою виклику методу MatrixComputationThread(); executor.execute(thread);
```

```
        }
        executor.shutdown();
        try {
            // Wait for all threads to finish
            while (!executor.isTerminated()) {
                Thread.sleep(100);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
//додайте свій код для виведення результатів
```

Варіанти завдань

Програма має обчислювати матриці, визначені користувачем.

Контрольні запитання

1. Опишіть використання методу sleep класу java.lang.Thread. Як можна достроково пробудити сплячий потік?
2. Опишіть діаграму станів потоку та методи, які приводять до зміни стану.
3. Дайте визначення критичної секції. Які механізми синхронного доступу до критичної секції пропонує Java?
4. Що таке об’єкт-монітор і як він використовується при синхронізації?
5. Що забезпечує модифікатор synchronized для методу?
6. Опишіть використання методів wait, notify та notifyAll класу Object для координації роботи потоків. Наведіть приклад їх використання.

Лабораторна робота №6.

Розробка геометричного методу Монте-Карло для обчислення площі фігури програми з використанням пулу потоків

Мета роботи: вивчити засоби Java для організації обчислень з використанням пулу потоків.

Теоретичні відомості

Створення потоку для кожного завдання вимагає значних комп'ютерних ресурсів. Певним виходом з такої ситуації є використання пулів потоків (Thread pool). Основна ідея використання пулу потоків полягає в наступному – коли програмі необхідно виконати деяку задачу, замість створення потоку для виконання цього завдання і передачі йому завдання, вона просто розміщує завдання у черзі (Job queue); – а один з працюючих в нескінченному циклі потоків з пулу потоків (Thread pool) вибирає завдання з черги і виконує його. Об'єкти, які реалізують такі функції, відомі як виконавці (Excecutors).

Пакет java.util.concurrent визначає три інтерфейси з функціональністю пулів потоків: Executor – простий інтерфейс, який підтримує запуск нових завдань. ExecutorService – успадковує Executor, додаючи функції, що забезпечують управління життєвим циклом як індивідуальних завдань, так і самого виконавця. ScheduledExecutorService – успадковує ExecutorService, даний інтерфейс додає можливість запускати відкладені завдання.

Для організації виконання завдання у потоці, що управляється об'єктом ExecutorService, необхідно: 1) реалізувати задачу в методі void run() об'єкта Runnable; 2) створити об'єкт Executor; 3) створити (отримати) робочий потік, виконуючий завдання шляхом виклику методу void execute(Runnable r), що додає завдання в чергу пулу потоків. Завдання буде заплановане і виконається доступним робочим потоком пулу.

Завдання

Обчислити співвідношення площин геометричних фігур (відповідно до варіанта) методом Монте-Карло з використанням полу потоків.

***Приклад*.** Обчислити співвідношення площин кола, яке вписано у квадрат заданої площі методом Монте-Карло з використанням полу потоків.

Пояснення: програма може складатися з наступних класів: MonteCarloCircleAreaSimulation створює пул потоків; виконує розпаралелювання симуляції шляхом ділення загальної кількості точок на рівні частини, і кожна частина обробляється окремим потоком у пулі потоків. Клас MonteCarloCircleAreaTask реалізує Runnable та представляє завдання, яке виконує кожен потік.

В методі run() генеруються випадкові точки всередині прямокутника та перевіряється, чи потрапляють вони всередину вписаного кола. Остаточна оцінка площі кола обчислюється на основі відношення точок всередині кола до загальної кількості точок у масштабі площі квадрата.

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicLong;
public class MonteCarloCircleAreaSimulation {
   public static void main(String[] args) {
      int totalPoints = 1000000;
      int numThreads = 4;
      ExecutorService executor = Executors.newFixedThreadPool(numThreads);
      AtomicLong insideCircleCounter = new AtomicLong(0);
      for (int i = 0; i < numThreads; i++) {
         executor.execute(new MonteCarloCircleAreaTask(totalPoints / numThreads, insideCircleCounter));
```

```
        }
        executor.shutdown();
        try {
            executor.awaitTermination(Long.MAX_VALUE,
TimeUnit.NANOSECONDS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        double rectangleArea = 2.0 * 2.0;
        double circleAreaEstimation = (insideCircleCounter.get() / (double)
totalPoints) * rectangleArea;
        System.out.println(circleAreaEstimation);
    }
}
class MonteCarloCircleAreaTask implements Runnable {
    private final int pointsInThread;
    private final AtomicLong insideCircleCounter;
    public MonteCarloCircleAreaTask(int pointsInThread, AtomicLong
insideCircleCounter) {
        this.pointsInThread = pointsInThread;
        this.insideCircleCounter = insideCircleCounter;
    }
    @Override
    public void run() {
        int insideCircle = 0;
        for (int i = 0; i < pointsInThread; i++) {
            double x = Math.random() * 2.0;
            double y = Math.random() * 2.0;
            if (isInsideCircle(x, y)) {
```

```
            insideCircle++;
          }
        }
        insideCircleCounter.addAndGet(insideCircle);
    }
    private boolean isInsideCircle(double x, double y) {
        return x * x + y * y <= 1.0;
    }
}
```

Варіанти

Варіант № 1

Обчислити співвідношення площин трикутника, який вписаний у квадрат заданої площі методом Монте-Карло з використанням полу потоків (задати максимальну кількістю потоків у полі =4).

Варіант № 2

Обчислити співвідношення площин трикутника, який вписаний у коло заданої площі методом Монте-Карло з використанням полу потоків (задати максимальну кількістю потоків у полі =5).

Варіант № 3

Обчислити співвідношення площин трикутника, який вписаний у паралелограм заданої площі методом Монте-Карло з використанням полу потоків (задати максимальну кількістю потоків у полі =6).

Варіант № 4

Обчислити співвідношення площин прямокутника, який вписаний у трапецію заданої площі методом Монте-Карло з використанням полу потоків (задати максимальну кількістю потоків у полі =3).

Варіант № 5

Обчислити співвідношення площин прямокутника, який вписаний у коло заданої площі методом Монте-Карло з використанням полу потоків (задати максимальну кількістю потоків у полі =4).

Варіант № 6

Обчислити співвідношення площин прямокутника, який вписаний у трикутник заданої площі методом Монте-Карло з використанням полу потоків (задати максимальну кількістю потоків у полі =6).

Варіант № 7

Обчислити співвідношення площин квадрата, який вписаний у коло заданої площі методом Монте-Карло з використанням полу потоків (задати максимальну кількістю потоків у полі =4).

Варіант № 8

Обчислити співвідношення площин квадрата, який вписаний у трикутник заданої площі методом Монте-Карло з використанням полу потоків (задати максимальну кількістю потоків у полі =8).

Варіант № 9

Обчислити співвідношення площин квадрата, який вписаний у трапецію заданої площі методом Монте-Карло з використанням полу потоків (задати максимальну кількістю потоків у полі =5).

Варіант № 10

Обчислити співвідношення площин трапеції, яка вписана у коло заданої площі методом Монте-Карло з використанням полу потоків (задати максимальну кількістю потоків у полі =5).

Контрольні запитання

1. Надайте загальну характеристику інструментам Java Concurrency API.
2. Назвіть переваги використання пулів потоків та опишіть їхню роботу.
3. Поясніть відміну завдань Runnable та Callable, що передаються до пулу потоків. Опишіть призначення та методи інтерфейсу Future.
4. Назвіть етапи організації виконання завдання Runnable у потоках, що управляються об’єктом ExecutorService.
5. Опишіть Методи інтерфейсу ExecutorService.

Лабораторна робота №7.

Розробка програми з організованим контролем доступу до спільного ресурсу

Мета роботи: вивчити засоби Java для організації контролю до спільного ресурсу.

Теоретичні відомості

Семафор (Semaphore) – об’єкт, який здійснює доступ потоків до ресурсу за допомогою лічильника дозволів на доступ. Об’єкт класу java.util.concurrent.Semaphore підтримує певну кількість дозволів на доступ до критичної секції, яка вказується як параметр конструктора під час створення семафора, наприклад, Semaphore semaphore = new Semaphore(5) – виклик конструктора семафору з одним параметром не забезпечує справедливість під час надання доступу потокам до критичної секції. Справедливе блокування (FairSync) – це коли потоки отримують блокування в тому порядку, в якому вони його запитували, несправедливе блокування (NonfairSync) може допускати «безлад» (barging), коли потік може отримати блокування раніше іншого, який запитував його першим. Під час передачі в конструктор другого параметра true буде забезпечуватися справедливе блокування (семафор буде використовувати FIFO чергу). Semaphore sem = new Semaphore(5, true).

Кожен виклик методу void acquire() з семафора в потоці зменшує на 1 кількість дозволів на доступ до критичної секції, кожен виклик методу void release() збільшує число дозволів на 1. Таким чином, кількість потоків, що одночасно виконують критичну секцію коду, обмежується кількістю дозволів семафора, а критична секція починається викликом acquire() і закінчується викликом release() (рис. 7.1). Методи семафора void acquire(int permits) та void release(int permits), які зменшують та збільшують одночасно кількість дозволів.

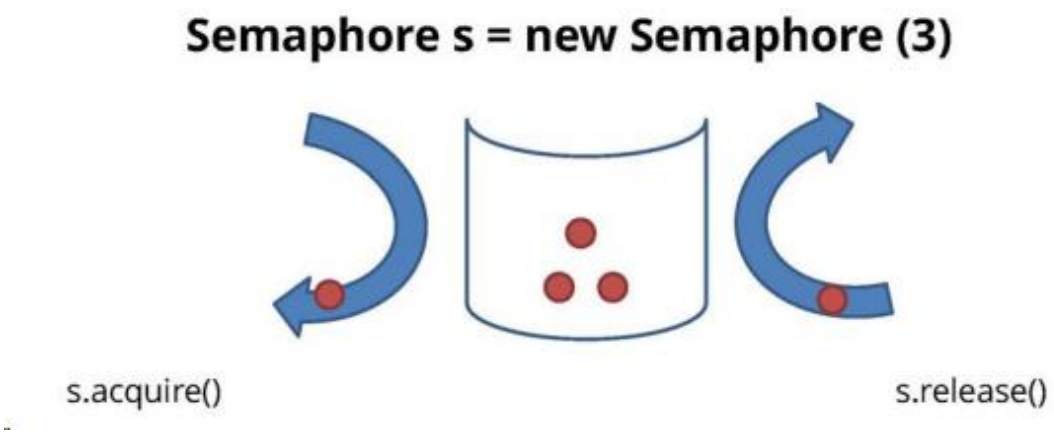

Рис. 7.1. Створення та робота семафора

Завдання

Прочитати файл заданого розміру в паралельних потоках з обмеженою кількістю одночасних звернень до файлу, яка задається об’єктом Семафор. Утримувати головний потік, доки всі потоки не закінчать виконання. Виводити на екран частину файлу, прочитану кожним потоком.

Приклад. Прочитати файл в 4-х паралельних потоках з обмеженою кількістю одночасних звернень до файлу, яка дорівнює 2-м.

Пояснення: програма може складатися з наступних класів:

Клас MultiThreadedFileReaderWithSemaphore визначає кількість потоків, реалізує метод readFromFileWithThreadsAndSemaphore, в якому об’єктом Семафор визначається кількість потоків, які отримають одночасний доступ до файлу. Визначається, як файл буде розділено для читання між потоками. Клас FileReadTask реалізує Callable, включає метод call(), в якому спочатку отримується доступ до файлу (semaphore.acquire()) і після зчитування повертається (semaphore.release()).

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.*;
public class MultiThreadedFileReaderWithSemaphore {
   private static Semaphore semaphore;
   public static void main(String[] args) {
      String filePath = "path/to/your/file.txt";
      int numThreads = 4;
      try {
         List<String> lines = readFromFileWithThreadsAndSemaphore(filePath, numThreads);
         for (String line : lines) {
            System.out.println(line);
         }
      } catch (InterruptedException | ExecutionException e) {
         e.printStackTrace();
      }
   }
   private static List<String> readFromFileWithThreadsAndSemaphore(String filePath, int numThreads)
         throws InterruptedException, ExecutionException {
      ExecutorService executorService = Executors.newFixedThreadPool(numThreads);
      semaphore = new Semaphore(numThreads);
```

```
            try (BufferedReader br = new BufferedReader(new
FileReader(filePath))) {
        List<Future<List<String>>> futures = new ArrayList<>();
        // Divide the file into chunks for each thread
        long fileSize = br.lines().count();
        long chunkSize = fileSize / numThreads;
        for (int i = 0; i < numThreads; i++) {
            long startLine = i * chunkSize;
            long endLine = (i == numThreads - 1) ? fileSize : (i + 1) * chunkSize;
            Callable<List<String>> task = new FileReadTask(filePath, startLine,
endLine);
            futures.add(executorService.submit(task));
        }
        List<String> result = new ArrayList<>();
        for (Future<List<String>> future : futures) {
            result.addAll(future.get());
        }
        return result;
    } finally {
        executorService.shutdown();
    }
  }

  private static class FileReadTask implements Callable<List<String>> {
    private final String filePath;
    private final long startLine;
    private final long endLine;

    public FileReadTask(String filePath, long startLine, long endLine) {
```

```
        this.filePath = filePath;
        this.startLine = startLine;
        this.endLine = endLine;
    }

    @Override
    public List<String> call() throws Exception {
        List<String> lines = new ArrayList<>();

        try {
            semaphore.acquire(); // Acquire a permit before reading
            try (BufferedReader br = new BufferedReader(new
FileReader(filePath))) {
                for (long i = 0; i < endLine; i++) {
                    String line = br.readLine();
                    if (i >= startLine) {
                        lines.add(line);
                    }
                }
            }
        } finally {
            semaphore.release(); // Release the semaphore after reading is done
        }

        return lines;
    }
  }
}
```

Варіант № 1

Прочитати файл в паралельних потоках, кількість який дорівнює 5 з обмеженою кількістю одночасних звернень до файлу, яка дорівнює 3-м. Необхідно утримувати головний потік, доки всі потоки не закінчать виконання, та виводити на екран прочитані потоком частини файлу.

Варіант № 2

Прочитати файл в паралельних потоках, кількість який дорівнює 6 з обмеженою кількістю одночасних звернень до файлу, яка дорівнює 2-м. Необхідно утримувати головний потік, доки всі потоки не закінчать виконання, та виводити на екран прочитані потоком частини файлу.

Варіант № 3

Прочитати файл в паралельних потоках, кількість який дорівнює 4 з обмеженою кількістю одночасних звернень до файлу, яка дорівнює 1-му. Необхідно утримувати головний потік, доки всі потоки не закінчать виконання, та виводити на екран прочитані потоком частини файлу.

Варіант № 4

Прочитати файл в паралельних потоках, кількість який дорівнює 7 з обмеженою кількістю одночасних звернень до файлу, яка дорівнює 4-м. Необхідно утримувати головний потік, доки всі потоки не закінчать виконання, та виводити на екран прочитані потоком частини файлу.

Варіант № 5

Прочитати файл в паралельних потоках, кількість який дорівнює 5 з обмеженою кількістю одночасних звернень до файлу, яка дорівнює 3-м. Необхідно утримувати головний потік, доки всі потоки не закінчать виконання, та виводити на екран прочитані потоком частини файлу.

Варіант № 6

Прочитати файл в паралельних потоках, кількість який дорівнює 8 з обмеженою кількістю одночасних звернень до файлу, яка дорівнює 3-м. Необхідно утримувати головний потік, доки всі потоки не закінчать виконання, та виводити на екран прочитані потоком частини файлу.

Варіант № 7

Прочитати файл в паралельних потоках, кількість який дорівнює 7 з обмеженою кількістю одночасних звернень до файлу, яка дорівнює 1-му. Необхідно утримувати головний потік, доки всі потоки не закінчать виконання, та виводити на екран прочитані потоком частини файлу.

Варіант № 8

Прочитати файл в паралельних потоках, кількість який дорівнює 8 з обмеженою кількістю одночасних звернень до файлу, яка дорівнює 3-м. Необхідно утримувати головний потік, доки всі потоки не закінчать виконання, та виводити на екран прочитані потоком частини файлу.

Варіант № 9

Прочитати файл в паралельних потоках, кількість який дорівнює 9 з обмеженою кількістю одночасних звернень до файлу, яка дорівнює 4-му. Необхідно утримувати головний потік, доки всі потоки не закінчать виконання, та виводити на екран прочитані потоком частини файлу.

Варіант № 10

Прочитати файл в паралельних потоках, кількість який дорівнює 10 з обмеженою кількістю одночасних звернень до файлу, яка дорівнює 4-м. Необхідно утримувати головний потік, доки всі потоки не закінчать виконання, та виводити на екран прочитані потоком частини файлу.

Контрольні запитання

1. Опишіть принцип роботи семафора (java.util.concurrent.Semaphore), параметри його конструктора та основні методи.
2. Що називають взаємовиключаючим семафором (Mutex-ом)?
3. Опишіть принцип роботи засувки зі зворотним відліком (java.util.concurrent.CountDownLatch), параметри його конструктора та основні методи.
4. Опишіть принцип роботи циклічного бар’єра (java.until.concurrent.CyclicBarrier), параметри його конструктора та основні методи.

Список літератури

1. Корочкін О.В. Паралельні та розподілені обчислення. Вибрані розділи : навч. посібник / О.В. Корочкін, О.В. Русанова. – Київ : КПІ ім. Ігоря Сікорського, 2020. – 123 с.
2. Коцовський В. М. Теорія паралельних обчислень : навчальний посібник / В. М. Коцовський. – Ужгород : ПП «АУТДОР-Шарк», 2021. – 188 с.
3. Лісовенко І.Д. «Паралельні та розподілені обчислення» : навчальний посібник / І.Д. Лісовенко, І.Д. Яковлєва. – Чернівці : ЧНУ, 2022. – 120 с.
4. Малашонок Г. І. Паралельні обчислення на розподіленій пам’яті : OpenMPI, Java, Math Partner : підручник / Г. І. Малашонок, А. А. Сідько. – Київ : НаУКМА, 2020. – 266 с.
5. Минайленко Р.М. Паралельні та розподілені обчислення : навч. посіб. – Кропивницький : Видавець Лисенко В. Ф., 2021. – 153 с.

Інформаційні ресурси в Інтернет

1. Java специфікація класу Циклічний бар’єр. – Режим доступу CyclicBarrier (Java Platform SE 8) (oracle.com)
2. Java специфікація класу Семафор. – Режим доступу Semaphore (Java Platform SE 8) (oracle.com)
3. Java специфікація станів потоку. – Режим доступу Thread.State (Java Platform SE 8) (oracle.com)
4. Java специфікація інтерфейсу Lock. – Режим доступу Lock (Java SE 11 & JDK 11) (oracle.com)
5. Java специфікація інтерфейсу Executor. – Режим доступу Executor (Java Platform SE 8) (oracle.com)
6. Java специфікація інтерфейсу Executor. – Режим доступу ExecutorService (Java Platform SE 8) (oracle.com)

ДЛЯ НОТАТОК

Навчально-методичне видання

ТЕХНОЛОГІЇ РОЗПОДІЛЕНИХ СИСТЕМ ТА ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

Методичні вказівки
до виконання лабораторних робіт
для підготовки здобувачів першого (бакалаврського) рівня вищої освіти
спеціальності 122 «Комп'ютерні науки»

Укладач СОЛОВЕЙ Ольга Леонідівна

Випусковий редактор *Л. С. Тавлуй*
Комп'ютерне верстання *К. А. Мавроді*

Підписано до друку 13.02.2025. Формат 60 x $84_{1/16}$
Ум. друк. арк. 2,79. Обл.-вид. арк. 3,0.
Електронний документ. Вид. № 190/III-24

Видавець і виготовлювач:
Київський національний університет будівництва і архітектури
Проспект Повітряних Сил, 31, Київ, Україна, 03037
Свідоцтво про внесення до Державного реєстру суб'єктів
видавничої справи ДК № 808 від 13.02.2002