Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №9

Виконала:

Студентка групи ІА-32

Павлюк В.О.

Київ 2025

1. Мета роботи

Навчитися реалізовувати потокобезпечні операції зі спільним станом у багатопотоковому середовищі, уникати взаємного блокування (deadlock) під час синхронізації, а також реалізувати задачу Producer–Consumer із використанням потокобезпечного кільцевого буфера.

2. Теоретичні відомості

У багатопотокових програмах кілька потоків можуть одночасно працювати зі спільними даними (shared state). Якщо доступ до спільного стану не синхронізувати, виникають race condition та некоректні результати (наприклад, “зникнення” грошей під час переказів між рахунками). У Java відсутня конструкція atomic для транзакцій, тому атомарність операцій досягається використанням механізмів синхронізації: synchronized, Lock/Condition, семафорів тощо.
Під час блокування кількох ресурсів одночасно існує ризик deadlock, коли потік A тримає блокування ресурсу 1 і чекає ресурс 2, а потік B тримає ресурс 2 і чекає ресурс 1. Один із надійних способів уникнути deadlock - завжди захоплювати блокування в однаковому глобальному порядку (наприклад, за ідентифікатором/хешем об’єкта).
Задача Producer-Consumer моделює систему, де одні потоки виробляють дані, а інші їх споживають. Для буфера діють класичні правила: якщо буфер порожній - споживач очікує; якщо буфер заповнений - виробник очікує. Для цього використовуються wait/notify або Lock/Condition. Кільцевий буфер (ring buffer) зберігає елементи у циклічній структурі, де індекси голови (head) і хвоста (tail) переходять на початок при виході за межі.

3.Хід роботи

Завдання 1. Переказ коштів у багатопотоковому середовищі

- Створено проєкт lab9 та пакет task1. Додано класи Account, Bank та BankTransferTest.
- У класі Account реалізовано баланс рахунку та окремий ReentrantLock для блокування рахунку під час переказу.
- У класі Bank реалізовано метод transfer(from, to, amount), який блокує обидва рахунки на час операції. Для уникнення deadlock використано правило глобального порядку захоплення блокувань за identityHashCode об’єктів. У випадку рівності хешів використано додатковий tieLock.

- Додатково реалізовано метод totalBalance(accounts) для підрахунку загальної суми грошей у банку з блокуванням усіх рахунків у стабільному порядку, щоб сума була коректною під час конкуренції потоків.
- Для тестування створено 200 рахунків із випадковими стартовими балансами. Обчислено загальну суму до виконання переказів.
- Запущено 4000 потоків, кожен із яких виконує переказ випадкової суми з випадкового рахунку на інший рахунок. Переказ не виконується, якщо на рахунку недостатньо коштів, щоб баланс не ставав негативним.
- Після завершення всіх потоків повторно обчислено загальну суму грошей у банку. Сума до і після переказів збігається, що підтверджує коректність синхронізації та відсутність втрати даних.

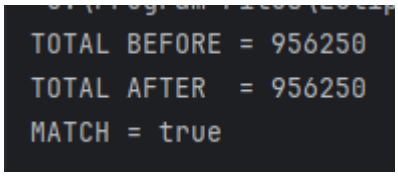

Рисунок 1 — Результат виконання завдання 1

Завдання 2. Producer–Consumer з кільцевим буфером

1. Створено пакет task2. Додано класи RingBuffer та ProducerConsumerMain.
2. Реалізовано потокобезпечний кільцевий буфер RingBuffer на основі замкненого у кільце зв’язного списку фіксованого розміру. Буфер використовує індекси head і tail та змінює їх по модулю розміру буфера, забезпечуючи циклічний доступ до елементів.
3. Для синхронізації використано ReentrantLock та умови notEmpty і notFull. Якщо буфер порожній, метод take() очікує появи елементів; якщо буфер повний, метод put() очікує звільнення місця.
4. У програмі ProducerConsumerMain створено два кільцеві буфери: перший для згенерованих повідомлень, другий для “перекладених” повідомлень.
5. Створено 5 потоків-демонів, які генерують рядки формату “Потік No ... згенерував повідомлення ...” та записують їх у перший буфер.
6. Створено 2 потоки-демони, які зчитують дані з першого буфера, формують новий рядок формату “Потік No ... переклав повідомлення ...” і записують у другий буфер.

7. Основний потік зчитує та друкує 100 повідомлень із другого буфера та завершує роботу. Оскільки допоміжні потоки є daemon, програма завершується автоматично після завершення main.

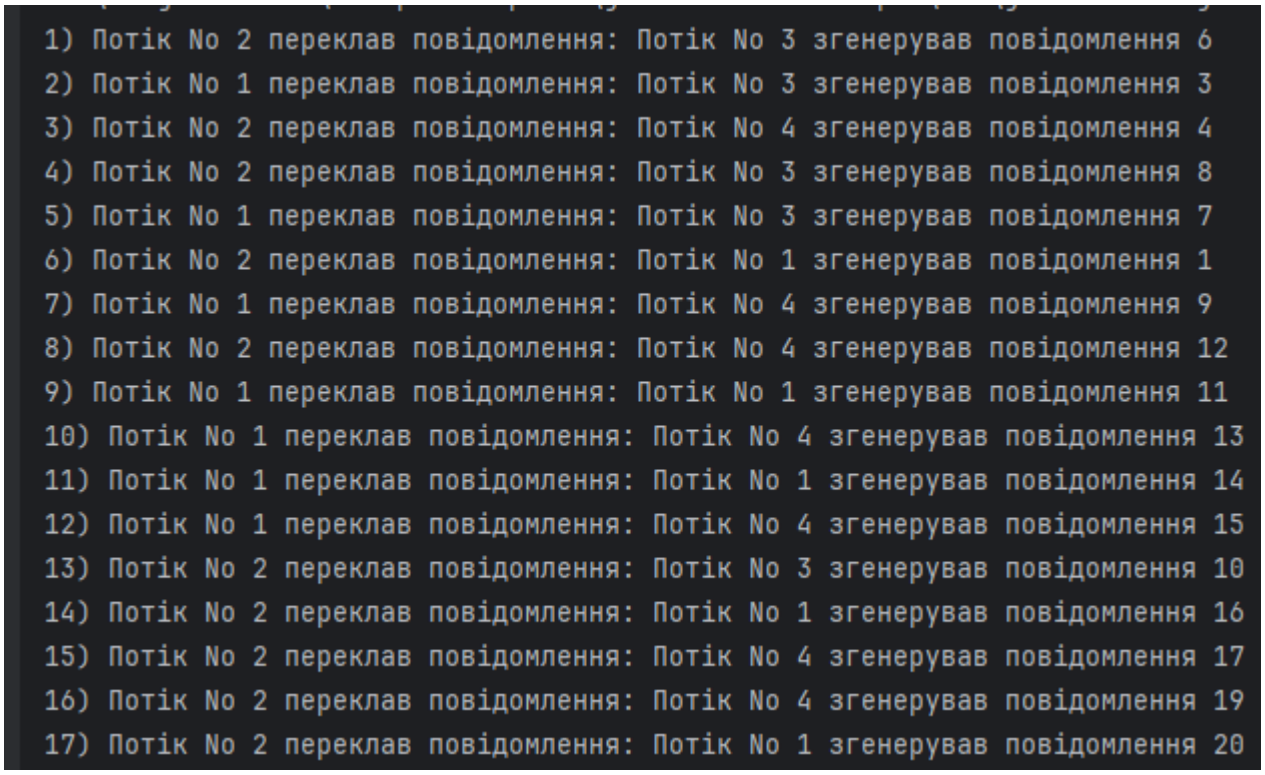

Рисунок 2 — Результат виконання завдання 2

Висновок: У ході лабораторної роботи було реалізовано два типові сценарії паралельного програмування. У першому завданні розроблено потокобезпечний механізм переказу коштів між рахунками зі спільним станом, забезпечено атомарність операції та уникнення deadlock за рахунок впорядкованого захоплення блокувань. Експериментальне тестування з великою кількістю потоків підтвердило, що загальна сума коштів у банку не змінюється до і після переказів.
У другому завданні реалізовано класичну задачу Producer–Consumer із використанням потокобезпечного кільцевого буфера. Буфер коректно обробляє ситуації порожнього та заповненого станів за допомогою механізмів очікування. Створені потоки-демони генерують і перетворюють повідомлення, а головний потік успішно зчитує та друкує 100 повідомлень, після чого програма коректно завершується.