Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

**Комп’ютерний практикум № 1**

з дисципліни «Технології паралельних обчислень»

**Тема:** «Розробка потоків та дослідження пріоритету запуску потоків»

|  |  |
| --- | --- |
| **Виконав:**  студент групи ІПз-11  Букрєєв Микита  Дата здачі \_\_\_\_\_\_\_\_  Захищено з балом \_\_\_\_\_\_\_ | **Перевірила:**  ас. кафедри ІПІ  Дифучина О.Ю. |

Київ 2024

**Завдання:**

1. Реалізуйте програму імітації руху більярдних кульок, в якій рух кожної кульки відтворюється в окремому потоці (див. презентацію «Створення та запуск потоків в java» та приклад). Спостерігайте роботу програми при збільшенні кількості кульок. Поясніть результати спостереження. Опишіть переваги потокової архітектури програм. **10 балів.**

Скріншот програми:

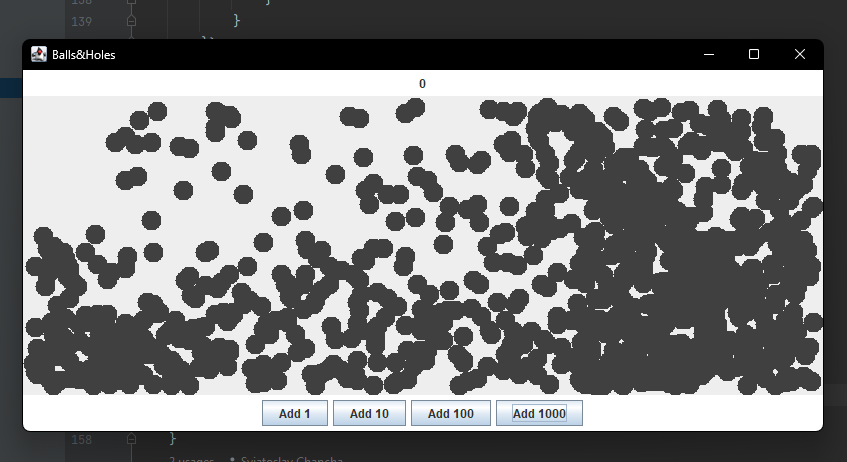


Рис. 1. Скріншот програми

Результати спостереження: Зі збільшенням кількості кульок зменшується їх швидкість.

Переваги потоків: потокова архітектура програм дозволяє виконувати обчислення паралельно, що дозволяє підвищити продуктивність. Це важливо для завдань, які вимагають великої кількості обчислень.

1. Модифікуйте програму так, щоб при потраплянні в «лузу» кульки зникали, а відповідний потік завершував свою роботу. Кількість кульок, яка потрапила в «лузу», має динамічно відображатись у текстовому полі інтерфейсу програми. **10 балів.**

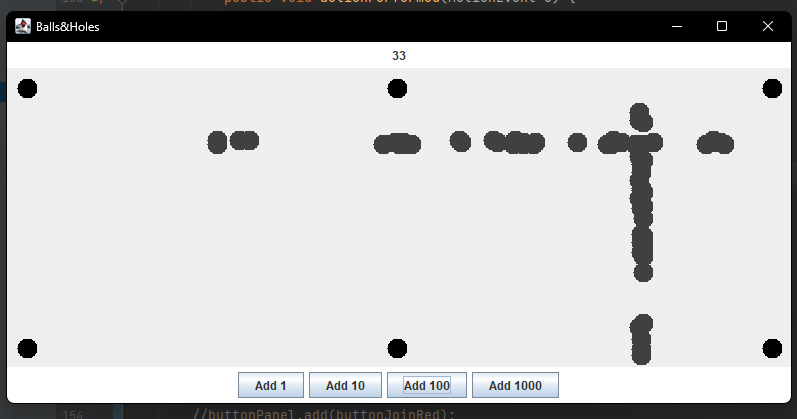


Рис. 2. Скріншот програми

1. Виконайте дослідження параметру priority потоку. Для цього модифікуйте програму «Більярдна кулька» так, щоб кульки червоного кольору створювались з вищим пріоритетом потоку, в якому вони виконують рух, ніж кульки синього кольору. Спостерігайте рух червоних та синіх кульок при збільшенні загальної кількості кульок. Проведіть такий експеримент. Створіть багато кульок синього кольору (з низьким пріоритетом) і одну червоного кольору, які починають рух в одному й тому ж самому місці більярдного стола, в одному й тому ж самому напрямку та з однаковою швидкістю. Спостерігайте рух кульки з більшим пріоритетом. Повторіть експеримент кілька разів, значно збільшуючи кожного разу кількість кульок синього кольору. Зробіть висновки про вплив пріоритету потоку на його роботу в залежності від загальної кількості потоків. **20 балів.**

Скріншот програми:

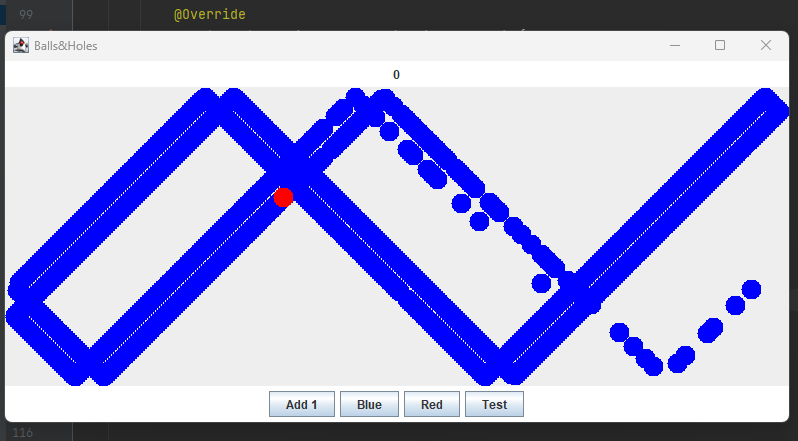


Рис. 3. Скріншот програми

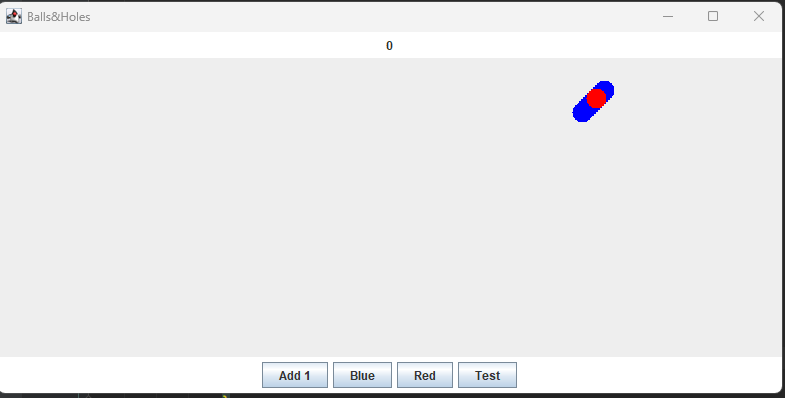


Рис. 4. Скріншот програми

Результати спостереження: Коли я викликав 100 синіх та 1 червону кульку, то вони загалом рухались однаково. Через якийсь час сині кульки почали відставати одне від одного, але червона кулька залишалась на місті. Коли я викликав 1000 синіх та 1 червону кульку, то вони починали набагато раніше за часом відставати одне від одного, і через якийсь час червона кулька виявилась найповільнішою і відстала від них. Результати я думаю пов`язані з тим, що чим більше в тебе потоків з меншим пріоритетом за потоків з більшим, то вони переймають більше ресурсів.

1. Побудуйте ілюстрацію для методу join() класу Thread з використанням руху більярдних кульок різного кольору. Поясніть результат, який спостерігається. **10 балів.**

Скріншот програми:



Рис. 5. Скріншот програми

Результати спостереження: наступний поток чекає кінець виконання діючого потоку, після чого і сам запускається. Основним методом у цьому очікуванні є саме join.

1. Створіть два потоки, один з яких виводить на консоль символ ‘-‘, а інший – символ ‘|’. Запустіть потоки в основній програмі так, щоб вони виводили свої символи в рядок. Виведіть на консоль 100 таких рядків. Поясніть виведений результат. **10 балів.** Використовуючи найпростіші методи управління потоками, добийтесь почергового виведення на консоль символів. **15 балів.**

Скріншот програми:

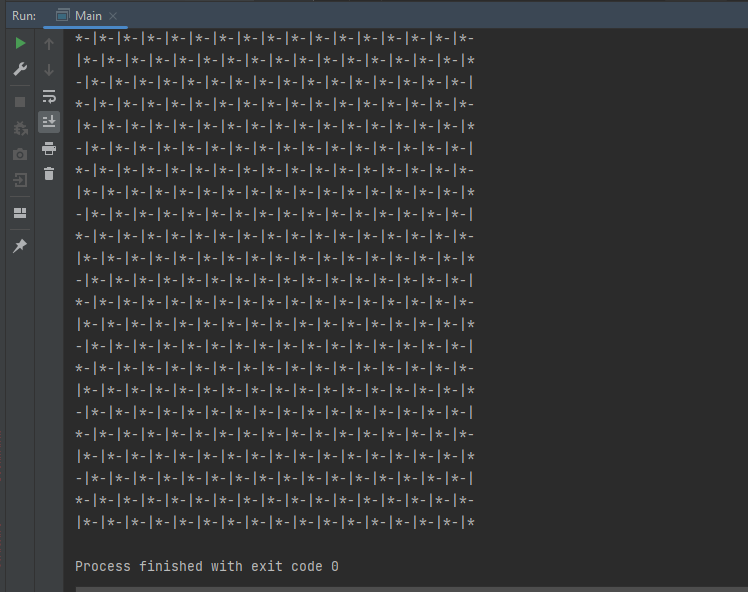


Рис. 6. Скріншот програми

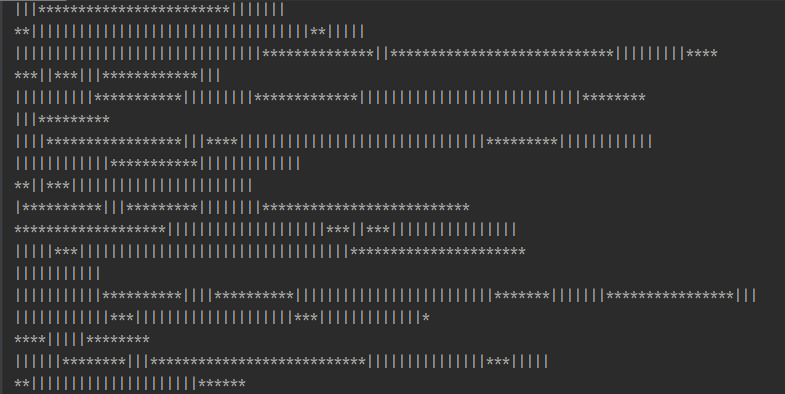


Рис. 7. Скріншот програми

Результат спостереження: Асинхронний спосіб виведення символів проходить хаотично і не дотримується зазначеної кількості символів у рядка. Також він не виводиться послідовно. Така ситуація відбувається оскільки виникає конфлікт при спільному доступі до консолі, конфлікт доступу до ресурсів, потоки не контактують один з одним та не впливають на рішення одне одного, що ми виправили з додаванням Syncer, який управляє потоками.

1. Створіть клас Counter з методами increment() та decrement(), які збільшують та зменшують значення лічильника відповідно. Створіть два потоки, один з яких збільшує 100000 разів значення лічильника, а інший – зменшує 100000 разів значення лічильника. Запустіть потоки на одночасне виконання. Спостерігайте останнє значення лічильника. Поясніть результат. **10 балів.** Використовуючи синхронізований доступ, добийтесь правильної роботи лічильника при одночасній роботі з ним двох і більше потоків. Опрацюйте використання таких способів синхронізації: синхронізований метод, синхронізований блок, блокування об’єкта. Порівняйте способи синхронізації. **15 балів.**

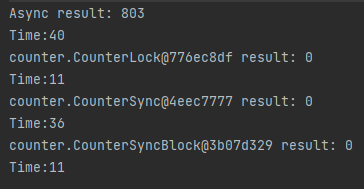


Рис. 8. Скріншот програми

Пояснення Async: Цей результат пов`язаний з так званими перегонами. Це коли два потоки здійснюють доступ до одного й того же об`єкта і «наступають один одному на п`яти». Тому результати можуть бути не очікуваними.

Порівняння способів синхронизації: Отже, ми маємо три різних способи синхронізації – синхронізований метод, синхронізований блок, блокування об`єкта. Використання ключового слова synchronized зменшує продуктивність програми, оскільки змушує потоки очікувати на доступ до ресурсу. Синхронізований блок може бути корисним, коли ми хочемо заблокувати доступ до певного розділу коду в середині методу, а не до всього методу в цілому. Це означає, що тільки один потік може виконувати цей блок одночасно, виключаючи інші потоки, які звертаються до цього ж об`єкта. Блокування об`єкта – коли потік звертається до методу, який заблокований для об`єкта, він буде зупинений до того моменту, поки інші потоки не закінчать свої дії з цим об`єктом. Як ми можемо помітити блокування об`єкту та синхронізований блок є більш швидкими аніж синхронізований метод.

Програмний код наведений в репозиторії:

<https://github.com/NikitaBukreyev/TPO>

**Висновки:**

Після виконання практичної роботи можна зробити висновок, що у багатьох випадках використання низькорівневих потоків може покращити швидкодію програми за рахунок паралельної обробки даних. Однак, ручна робота з кожним потоком може бути витратною за часом та написанням коду. Також, пріоритет потоків може змінювати характер виконання програми, але ці зміни не завжди передбачувані та відтворювані. Щоб отримати доступ до даних потрібно використовувати численні методи синхронізації, що є обов'язковим для правильної роботи програми з багатопотоковими додатками.