Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 1 з дисципліни
«Теорія паралельних обчислень»

«Розробка потоків та дослідження пріоритету запуску потоків»

Виконав(ла) *ІП-14 Сергієнко Ю. В.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Дифучина О. Ю.* _____
(прізвище, ім'я, по батькові)

Київ 2024

Комп’ютерний практикум 1

Тема: Розробка потоків та дослідження пріоритету запуску потоків.

Виконання:

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.**

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.**

Для виконанання даного завдання було додано можливість починати роботу потока на натискання кнопки Start. Щоб завершити роботу потоку, необхідно дочекатись проходження 10000 ітерацій або треба щоб кулька потрапила у лунку. Якщо кулька потрапила у лунку, спрацьовує вихід з циклу ітерацій за допомогою *break* та потік життєвий цикл потоку завершується. Також оновляється поле к-сті кульок у лунках (рисунок 1).

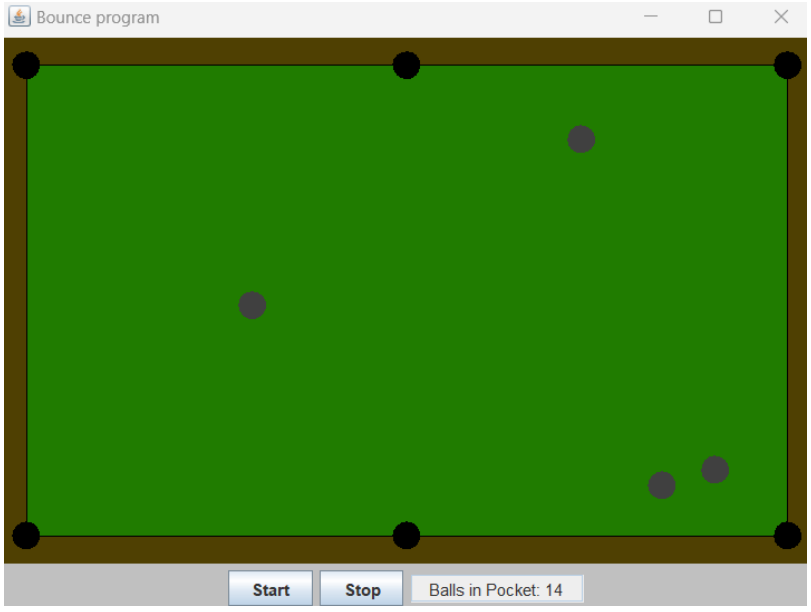

Рисунок 1 – Додано лунки та текстове поле

2. 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.**

Було модифіковано основну програму, а саме: на клік кнопки Start тепер створюється багато кульок, одна з яких – червоного кольору з високим пріоритетом, інші – сині з низьким пріоритетом. Протестуємо з 100 кульками (рисунок 2).

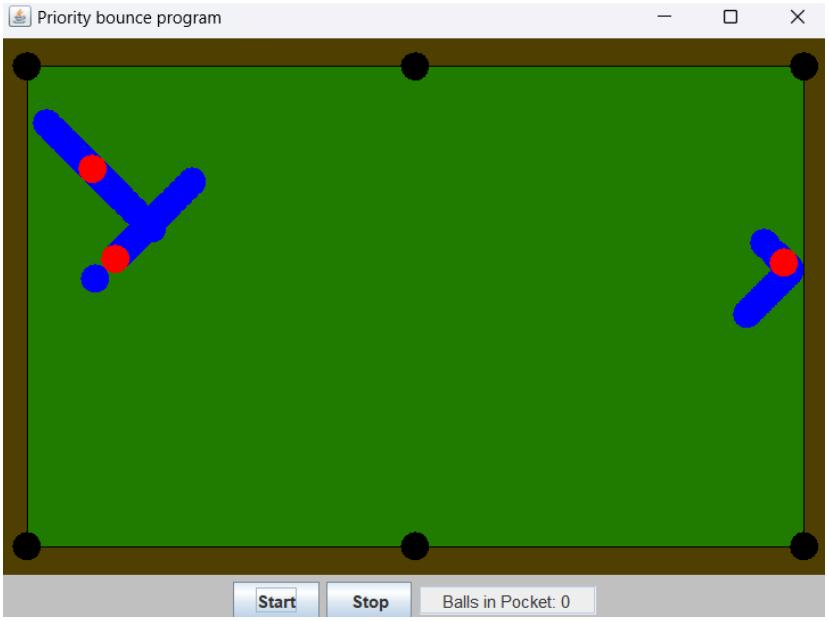

Рисунок 2 – Модифікована програма, 100 кульок

За логікою, червона кулька має переганяти сині, але бачимо, що це не так. Фактично, високий пріоритет не гарантує його першочергового

виконання. До того ж, потоки з меншим пріоритетом наразі переважають і, таким чином, домінують при захоплені ресурсів системи. Протестуємо з 500 кульками (рисунок 3).

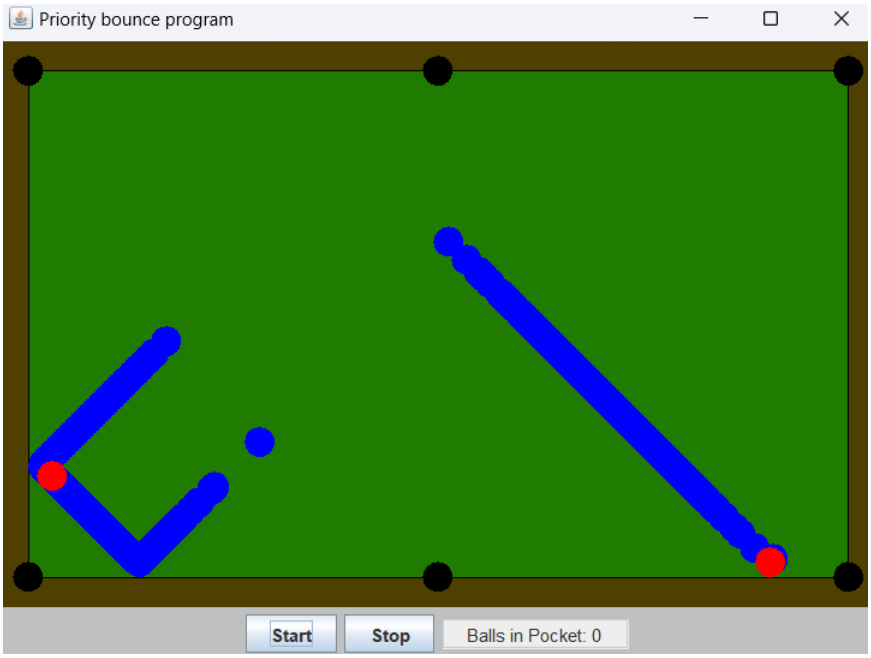

Рисунок 3 – Модифікована програма, 500 кульок

Як бачимо, пріоритет потоку не може гарантувати першості.

3. Побудуйте ілюстрацію методу join() класу Thread через взаємодію потоків, що відтворюють рух більярдних кульок різного кольору. Поясніть результат, який спостерігається. **10 балів.**

Модифікуємо програму на створення однієї кульки по кліку (одного потоку). Щоб досягнути правильної роботи програми, поточному потоку необхідно дочікуватись виконання минулого. Для цього, при його створенні будемо передавати поток минулої кульки та дочікуватись його виконання за допомогою методу *join()*. Результат роботи програми відображено на рисунку 4.

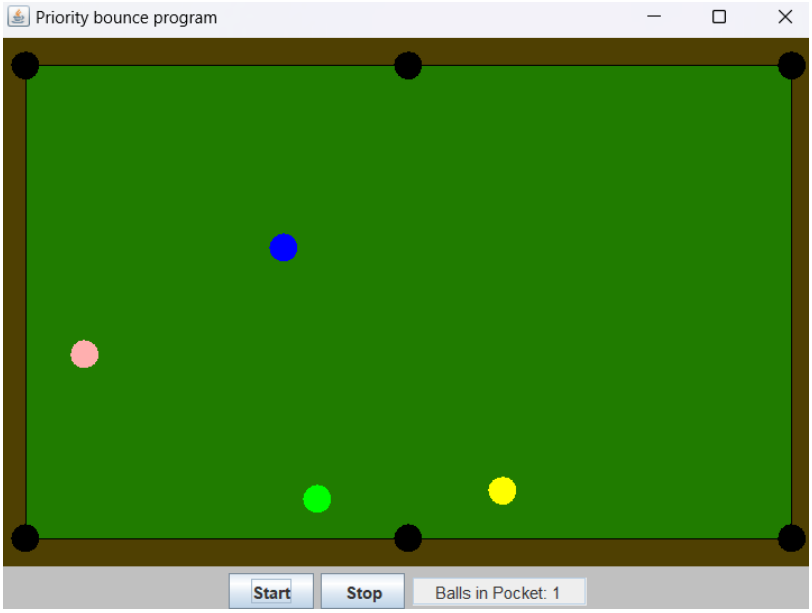

Рисунок 4 – Робота програми із методом join()

4. 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**.

Спробуємо вивести символи у консоль. Як бачимо, замість «-|» рядків було виведено набір невпорядкованих (як необхідно) символів. Це відбувається через те, що робота потоків не синхронізована, а, отже, перший поток не дочікується виконання другого і виводить свій символ випадково (рисунок 5).

Рисунок 5 – Консоль виконання несинхронізованих потоків

Для синхронізації потоків використаємо синхронізований блок та стандартний локер Object lock. Вивід буде розпочинати один з потоків, інший – чекає результату першого та запускається після виклику *lock.notify()*. Результат можна побачити на рисунку 6.

Рисунок 6 – Консоль виконання синхронізованих потоків

5. 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**.

Було створено два потоки та об'єкт класу PlainCounter для одночасного доступу до даних. Запустимо потоки та відобразимо результат у консоль. Бачимо, що фінальне число не дорівнює 0, адже під час виконання дій одночасно, counter використовувало два потоки та перезаписували значення лічильника (рисунок 7).

```
Plain counting:
RESULT: 3950
RESULT: -42750
```

Рисунок 7 – Результат виконання першої програми лічильника

Синхронізуємо виконання потоків за допомогою синхронізованих методів інкременту та декременту (рисунок 8), сихронізованого блоку (рисунок 9) та за допомогою блокування об'єка (рисунок 10).

```
Synchronization using synchronized methods:
RESULT: 0
```

Рисунок 8 – Результат виконання програми (синх. метод)

```
Synchronization using synchronized blocks:
RESULT: 0
```

Рисунок 9 – Результат виконання програми (синх. блок)

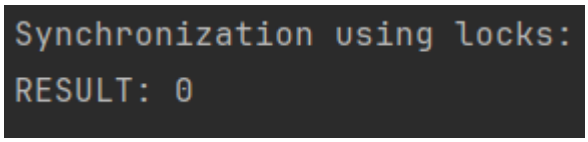

Рисунок 10 – Результат виконання програми (блокування об’єкта)

Висновок

Під час виконання даного комп’ютерного практикуму я набув знань та навичок щодо застосування потоків на практиці, їх особливостей та параметрів. Я дослідив їх несинхронізовану роботу та параметр пріоритетного запуску. Також було досліджено роботу синхронізованих потоків за допомогою синхронізованих блоків, методів та об’єктів блокування.

Код програми доступний на Github.