**Лабораторна робота №3**

Шевченко Н. В.

**Тема:** Розширені засоби синхронізації потоків (семафори).

**Мета:** Познайомитись із використанням семафорів як засобів синхронізації потоків у різних мовах програмування.

**Ada**

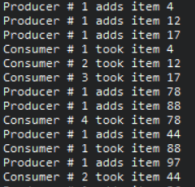
Цей код на мові програмування Ada моделює систему з постачальниками та споживачами, які взаємодіють через спільне сховище, використовуючи семафори та контейнери Ada.

Програма починається з оголошення типу RandRange, який визначає діапазон для генерації випадкових чисел. Далі визначується protected тип ProductionControl, який відповідає за контроль виробництва і споживання. У тілі цього protected типу реалізовані процедури для встановлення загальної кількості елементів, зменшення кількості вироблених та спожитих елементів, а також функції для перевірки закінчення виробництва та споживання.

Далі оголошені семафори Access\_Control, Full\_Storage, та Empty\_Storage, які контролюють доступ до сховища, заповненість та порожнечу сховища відповідно. Визначені типи завдань Supplier та Receiver, які відповідають за поведінку постачальників та споживачів відповідно. Кожен з них містить entry для ініціалізації та task body, де реалізовано їхню роботу.

В тілі Supplier генеруються випадкові числа та додаються до сховища, поки виробництво не завершиться. В тілі Receiver беруться елементи з сховища та виводяться у консоль, поки споживання не завершиться.

У блоку begin оголошуються масиви постачальників та споживачів, які потім ініціалізуються та запускаються.



**Java**

Цей Java-код моделює систему з виробниками та споживачами, які взаємодіють через спільне сховище.

У методі main класу Main спочатку визначаються параметри: ємність сховища capacity, загальна кількість елементів totalItems, кількість виробників producersCount та кількість споживачів consumersCount. Потім створюється об'єкт класу Storage, який представляє сховище, з вказаною ємністю.

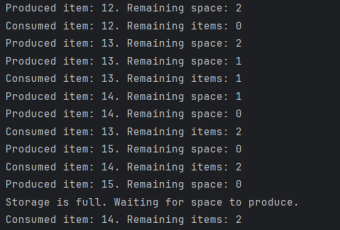
У наступному циклі створюються та запускаються потоки виробників та споживачів. Кожен виробник створюється за допомогою об'єкта класу Producer, передаючи йому об'єкт сховища, загальну кількість елементів та ідентифікатор виробника. Аналогічно, кожен споживач створюється за допомогою об'єкта класу Consumer, передаючи йому об'єкт сховища та загальну кількість елементів.

Клас Producer реалізує інтерфейс Runnable і містить логіку виробництва елементів. У його методі run відбувається створення унікального елемента для кожного виробника та його додавання до сховища.

Клас Consumer також реалізує інтерфейс Runnable і містить логіку споживання елементів. У його методі run елементи витягаються з сховища.

Клас Storage відповідає за сховище. В ньому реалізовані методи produce та consume, які синхронізовані, тобто забезпечують безпечний доступ до спільного ресурсу у випадку багатопотоковості. Клас також містить поле buffer, яке представляє чергу для зберігання елементів, та поле capacity, яке визначає ємність сховища.

Коли сховище повне, виробники застосовують метод wait(), щоб зачекати, поки звільниться місце для виробництва. Коли сховище порожнє, споживачі також викликають метод wait(), щоб зачекати, поки з'являться елементи для споживання. Після додавання елемента виробник викликає notifyAll(), щоб повідомити всіх чекаючих про те, що можна продовжити виконання. Аналогічно, після витягнення елемента споживач викликає notifyAll(), щоб повідомити всіх чекаючих виробників, що з'явилося місце для виробництва.



**Висновок**

У цій лабораторній роботі було розглянуто використання семафорів як засобів синхронізації потоків у мовах програмування Ada та Java. Наведені приклади кодів моделюють системи з постачальниками/виробниками та споживачами, які взаємодіють через спільне сховище.

В Ada використовуються захищені типи, семафори та контейнери для управління доступом до сховища та синхронізації потоків. У Java застосовуються методи wait() та notifyAll() для очікування та повідомлення потоків про зміни у сховищі.

Обидва приклади демонструють ефективне використання семафорів для забезпечення безпечного доступу до спільного ресурсу в умовах багатопотоковості, що є важливим аспектом у розробці паралельних програм.