Гаврилюк А.В.

Лабораторна робота №4

Тема: Взаємне блокування потоків (задача про філософів, що обідають).

Мета: Познайомитись з основними причинами виникнення взаємного блокування паралельних потоків, розібрати способи уникнення блокування потоків на прикладі задачі про філософів, що обідають.

Теоретичні відомості:

Взаємне блокування (deadlock) виникає при виконанні паралельних програм, якщо одночасно виконуються всі умови:

1. Взаємне виключення (mutual exclusion) – ресурси використовуються ексклюзивно.
2. Утримання та очікування (hold and wait) – потоки утримують ресурси та чекають нових.
3. Відсутність примусового звільнення (no preemption) – ресурси не можуть бути вилучені примусово.
4. Циклічне очікування (circular wait) – існує цикл потоків, кожен чекає ресурс, який утримує наступний.

Хід роботи:

Для задачі п’ятьох філософів використано п’ять потоків (Philosopher), кожен з яких по черзі думає та їсть. Для їжі філософу необхідно взяти дві вилки (семафори), які лежать між ним та сусідами.

Вирішення проблеми deadlock: з метою уникнення тупика у реалізації:

- Для перших чотирьох філософів вилки беруться у порядку: ліву, потім праву.
- Для останнього філософа порядок навпаки: спочатку права вилка, потім ліва.

Це розриває циклічне очікування, що усуває можливість deadlock.

Результати тестування

- Програма була запущена 15 разів.
- В усіх запусках всі філософи успішно завершили всі свої прийоми їжі.
- Відсутні ситуації взаємного блокування.
- Вивід програми чітко показує цикли «думання» та «їжі» кожного філософа.

- Завершення роботи відбувається після того, як всі філософи з’їли 5 разів.

Консольний вивыд Java та C#:

```
Philosopher 1 is thinking.
Philosopher 3 is thinking.
Philosopher 2 is thinking.
Philosopher 4 is thinking.
Philosopher 0 is thinking.
Philosopher 3 is eating (meal 1).
Philosopher 0 is eating (meal 1).
Philosopher 3 is thinking.
Philosopher 2 is eating (meal 1).
Philosopher 0 is thinking.
Philosopher 4 is eating (meal 1).
Philosopher 2 is thinking.
Philosopher 1 is eating (meal 1).
Philosopher 4 is thinking.
Philosopher 3 is eating (meal 2).
Philosopher 1 is thinking.
Philosopher 0 is eating (meal 2).
Philosopher 3 is thinking.
Philosopher 2 is eating (meal 2).
Philosopher 2 is thinking.
Philosopher 3 is eating (meal 3).
Philosopher 3 is thinking.
Philosopher 3 is eating (meal 4).
Philosopher 3 is thinking.
Philosopher 0 is thinking.
Philosopher 1 is eating (meal 2).
Philosopher 4 is eating (meal 2).
Philosopher 1 is thinking.
Philosopher 4 is thinking.
Philosopher 3 is eating (meal 5).
Philosopher 4 is eating (meal 3).
Philosopher 2 is eating (meal 3).
Philosopher 4 is thinking.
Philosopher 2 is thinking.
Philosopher 1 is eating (meal 3).
Philosopher 1 is thinking.
Philosopher 0 is eating (meal 3).
Philosopher 2 is eating (meal 4).
Philosopher 2 is thinking.
Philosopher 0 is thinking.
Philosopher 4 is eating (meal 4).
Philosopher 1 is eating (meal 4).
Philosopher 1 is thinking.
Philosopher 2 is eating (meal 5).
Philosopher 4 is thinking.
Philosopher 0 is eating (meal 4).
Philosopher 0 is thinking.
Philosopher 4 is eating (meal 5).
Philosopher 1 is eating (meal 5).
Philosopher 0 is eating (meal 5).
Dinner is over.
```

```
Philosopher 3 is thinking.
Philosopher 2 is thinking.
Philosopher 1 is thinking.
Philosopher 4 is thinking.
Philosopher 0 is thinking.
Philosopher 4 is eating (meal 1).
Philosopher 1 is eating (meal 1).
Philosopher 4 is thinking.
Philosopher 3 is eating (meal 1).
Philosopher 1 is thinking.
Philosopher 0 is eating (meal 1).
Philosopher 0 is thinking.
Philosopher 3 is thinking.
Philosopher 2 is eating (meal 1).
Philosopher 4 is eating (meal 2).
Philosopher 4 is thinking.
Philosopher 2 is thinking.
Philosopher 1 is eating (meal 2).
Philosopher 3 is eating (meal 2).
Philosopher 3 is thinking.
Philosopher 0 is eating (meal 2).
Philosopher 1 is thinking.
Philosopher 2 is eating (meal 2).
Philosopher 3 is eating (meal 3).
Philosopher 1 is eating (meal 3).
Philosopher 0 is thinking.
Philosopher 2 is thinking.
Philosopher 1 is thinking.
Philosopher 4 is eating (meal 3).
Philosopher 3 is thinking.
Philosopher 2 is eating (meal 3).
Philosopher 2 is thinking.
Philosopher 1 is eating (meal 4).
Philosopher 3 is eating (meal 4).
Philosopher 4 is thinking.
Philosopher 0 is eating (meal 3).
Philosopher 1 is thinking.
Philosopher 3 is thinking.
Philosopher 2 is eating (meal 4).
Philosopher 2 is thinking.
Philosopher 3 is eating (meal 5).
Philosopher 0 is thinking.
Philosopher 1 is eating (meal 5).
Philosopher 4 is eating (meal 4).
Philosopher 2 is eating (meal 5).
Philosopher 4 is thinking.
Philosopher 0 is eating (meal 4).
Philosopher 0 is thinking.
Philosopher 4 is eating (meal 5).
Philosopher 0 is eating (meal 5).
Dinner is over.
```

Висновок:

У результаті лабораторної роботи було досліджено проблему взаємного блокування у багатопотокових програмах. На прикладі класичної задачі про філософів, що обідають, продемонстровано умови виникнення deadlock та один з ефективних способів його усунення — зміна порядку захоплення ресурсів у останнього потоку. Реалізація задачі показала, що запропонований підхід дозволяє уникнути тупикових ситуацій та забезпечує коректне завершення програми.

Посилання на репозиторій з кодом програми:

https://github.com/AnnaHavryliuk4/ParallelProcessesCourse/tree/main/lab4_