Проблема обідаючих філософів

Виконав Пристайчук Дмитро, ММШІ-1

Задача 1: N=5 з використанням одного лічильного семафора

Структури даних

```
class Philosopher:
    id: Integer                 // Ідентифікатор філософа
    room: Semaphore             // Лічильний семафор для обмеження доступу
```

Алгоритм для філософа

```
Philosopher_Process(id, room):
    while true:
        Think(id)               // Філософ думає

        Wait(room)              // Спроба увійти в кімнату

        PickUpForks(id)         // Взяти обидві виделки
        Eat(id)                 // Філософ їсть
        PutDownForks(id)        // Покласти обидві виделки

        Signal(room)            // Вийти з кімнати
```

Ініціалізація

```
Initialize_Dining(n):
    room = new Semaphore(n-1)   // Ініціалізація семафора значенням n-1

    // Створення потоків філософів
    for i = 0 to n-1:
        Create_Thread(Philosopher_Process, i, room)
```

Пояснення рішення

Цей підхід використовує один лічильний семафор, ініціалізований значенням (n-1), щоб обмежити кількість філософів, які можуть одночасно намагатися взяти виделки. Оскільки для взаємного блокування (deadlock) усі n філософів повинні одночасно тримати по одній виделці, обмеження кількості філософів за столом до (n-1) робить взаємне блокування структурно неможливим.

Приклад виконання

```
Philosopher 0 is thinking
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 0 is hungry
Philosopher 0 has entered the dining room
Philosopher 0 picked up both forks
Philosopher 0 is eating
Philosopher 3 is hungry
Philosopher 3 has entered the dining room
Philosopher 3 picked up both forks
Philosopher 3 is eating
Philosopher 1 is hungry
Philosopher 1 has entered the dining room
Philosopher 1 picked up both forks
Philosopher 1 is eating
Philosopher 2 is hungry
Philosopher 2 has entered the dining room
Philosopher 2 picked up both forks
Philosopher 2 is eating
Philosopher 4 is hungry
Philosopher 0 put down both forks
Philosopher 0 has left the dining room
Philosopher 4 has entered the dining room
Philosopher 4 picked up both forks
Philosopher 4 is eating
Philosopher 0 is thinking
```

Задача 2: N=5 з мьютексом та п'ятьма семафорами виделок

Структури даних

```
class Philosopher:
    id: Integer                 // Ідентифікатор філософа
    mutex: Mutex                // М'ютекс для захисту доступу до виделок
    forks: Array of Semaphores  // Масив семафорів для кожної виделки
```

Алгоритм для філософа

```
Philosopher_Process(id, mutex, forks):
    left_fork = id
    right_fork = (id + 1) % n
```

```
    while true:
        Think(id)              // Філософ думає

        Lock(mutex)            // Захист критичної секції
        Wait(forks[left_fork])  // Взяти ліву виделку
        Wait(forks[right_fork]) // Взяти праву виделку
        Unlock(mutex)          // Звільнити мьютекс

        Eat(id)                // Філософ їсть

        Signal(forks[left_fork])  // Покласти ліву виделку
        Signal(forks[right_fork]) // Покласти праву виделку
```

Ініціалізація

```
Initialize_Dining(n):
    mutex = new Mutex()
    forks = Array of n new Semaphores(1)

    // Створення потоків філософів
    for i = 0 to n-1:
        Create_Thread(Philosopher_Process, i, mutex, forks)
```

Пояснення рішення

Цей підхід використовує мьютекс для захисту критичної секції взяття виделок і окремий семафор для кожної виделки. Мьютекс гарантує, що лише один філософ може намагатися взяти виделки одночасно, що запобігає ситуації, коли кожен філософ тримає одну виделку і чекає на іншу.

Приклад виконання

```
Philosopher 0 is thinking
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 2 is hungry
Philosopher 2 picked up forks 2 and 3
Philosopher 2 is eating
Philosopher 0 is hungry
Philosopher 3 is hungry
Philosopher 4 is hungry
Philosopher 1 is hungry
Philosopher 2 put down forks 2 and 3
Philosopher 0 picked up forks 0 and 1
Philosopher 0 is eating
Philosopher 2 is thinking
```

```
Philosopher 3 picked up forks 3 and 4
Philosopher 3 is eating
Philosopher 0 put down forks 0 and 1
Philosopher 4 picked up forks 4 and 0
Philosopher 4 is eating
Philosopher 0 is thinking
Philosopher 1 picked up forks 1 and 2
Philosopher 1 is eating
```

Задача 3: N=4 філософів та 5 паличок (виделок)

Доведення відсутності взаємного блокування для 4 філософів з 5 паличками

Твердження

У системі з 4 філософами і 5 паличками (виделками) взаємне блокування неможливе при стандартному алгоритмі обідаючих філософів.

Доведення від супротивного

1. **Припустимо, що відбувається взаємне блокування**: Це означає, що всі 4 філософи одночасно тримають по одній паличці і чекають на другу.
2. **Підрахунок ресурсів у стані блокування**:
    - Кількість філософів у блокуванні = 4
    - Кількість утримуваних паличок = 4 (по одній на філософа)
    - Загальна кількість паличок = 5
3. **Протиріччя**:
    - Якщо 4 палички утримуються (по одній кожним філософом), то має залишитися 1 вільна паличка.
    - Ця вільна паличка повинна бути суміжною з щонайменше одним філософом.
    - Цей філософ може взяти другу паличку і почати їсти.
    - Після їжі філософ звільняє обидві палички, роблячи їх доступними для сусідніх філософів.
    - Це руйнує умову взаємного блокування.
4. **Формальний аналіз ресурсів**:
    - Нехай C - кількість паличок (5)
    - Нехай P - кількість філософів (4)
    - Для взаємного блокування: P паличок повинні утримуватися одночасно
    - Якщо C > P (у нашому випадку 5 > 4), то принаймні одна паличка повинна залишитися доступною
    - Ця доступна паличка запобігає умові циклічного очікування

Висновок

Маючи більше паличок (5), ніж філософів (4), ми структурно усуваємо можливість взаємного блокування в проблемі обідаючих філософів. Це відбувається тому, що додаткова паличка гарантує, що принаймні один філософ завжди може завершити процес їжі, розриваючи потенційне циклічне очікування.

Приклад виконання (для 4 філософів і 5 паличок)

Початковий стан:

```
chopsticks[0]: available
chopsticks[1]: available
chopsticks[2]: available
chopsticks[3]: available
chopsticks[4]: available
philosophers[0]: thinking
philosophers[1]: thinking
philosophers[2]: thinking
philosophers[3]: thinking
```

Припустимо найгірший сценарій, коли кожен філософ бере одну паличку:

```
philosophers[0] бере chopsticks[0]
philosophers[1] бере chopsticks[1]
philosophers[2] бере chopsticks[2]
philosophers[3] бере chopsticks[3]
```

Результат:

```
chopsticks[0]: held by philosophers[0]
chopsticks[1]: held by philosophers[1]
chopsticks[2]: held by philosophers[2]
chopsticks[3]: held by philosophers[3]
chopsticks[4]: available (!)
```

Палиця chopsticks[4] залишається доступною, і philosopher[0] може взяти її (оскільки це його права паличка) та почати їсти. Після їжі він звільнить обидві палички, дозволяючи іншим філософам продовжити.