Programowanie współbieżne

Lista zadań nr 4

Na ćwiczenia 4. listopada 2022

Zadanie 1. Pokaż, że poniższa implementacja zamka dla n wątków spełnia warunek wzajemnego wykluczania. Uzasadnij i wykorzystaj następujący fakt: każdy wątek wykonujący metodę **lock()** znajduje się na jednym z n-1 poziomów, z których ostatni oznacza zajęcie zamka. Na poziomie n - i znajduje się jednocześnie co najwyżej i wątków.

Wskazówka: The Art of Multiprocessor Programming 2e, rozdział 2.5.

```
class Filter implements Lock {
    int[] level;
    int[] victim;
    public Filter(int n) {
        level = new int[n];
        victim = new int[n]; // use 1..n-1
        for (int i = 0; i < n; i++) {
            level[i] = 0;
        }
    }
    public void lock() {
        int me = ThreadID.get(); // returns 0..n-1
        for (int i = 1; i < n; i++) { // attempt to enter level i
            level[me] = i;
            victim[i] = me;
            // spin while conflicts exist
            while (( ∃ k != me) (level[k] >= i && victim[i] == me)) {};
        }
    }
    public void unlock() {
        int me = ThreadID.get();
        level[me] = 0;
    }
}
```

Zadanie 2. Pokaż, że metoda **lock()** klasy Filter spełnia warunek niezagłodzenia. Wywnioskuj stąd, że spełnia również warunek niezakleszczenia.

Zadanie 3. Przypomnij, co to znaczy że algorytm ma własność r-ograniczonego czekania (ang. *r-Bounded Waiting*). Pokaż, że algorytm Petersena ma własność 0-ograniczonego czekania, tzn, że jest FCFS (*First Come First Served*).

Zadanie 4. Pokaż, że nie istnieje taka stała r, że operacja przejścia na kolejny poziom w metodzie **lock()** klasy Filter ma własność r-ograniczonego czekania. Za sekcję wejściową przyjmij instrukcje ustalające poziom i ofiarę. Dlaczego ten wynik nie jest sprzeczny z własnością niezagłodzenia?

Zadanie 5. Dlaczego w definicji r-ograniczonego czekania musimy definiować sekcję wejściową jako złożoną z kilku pierwszych instrukcji, a nie jako pierwszą instrukcję metody? Przeprowadź dowód nie wprost rozważając przypadki, gdy pierwsza instrukcja jest: 1. odczytem tej samej komórki lub różnych komórek pamięci w zależności od wątku, 2. zapisem do różnych komórek, 3. zapisem do tej samej komórki.

Wskazówka: aby wypracować intuicje co do dwóch ostatnich przypadków, rozważ algorytm Petersena (punkt 2.) oraz jego modyfikację, gdzie najpierw następuje ustalenie ofiary, a dopiero po tym flagi (punkt 3). Możesz przyjąć, że te instrukcje używają ThreadID.get() do ustalenia wartości i.

Definicja 1. Formalną definicję **linearyzacji** można streścić w dwóch punktach:

1. Ciąg wywołania metod w programie współbieżnym powinien mieć taki efekt, jak gdyby te metody zostały wykonane w pewnym sekwencyjnym porządku, jedna po drugiej.
2. Efekt każdej metody w programie współbieżnym powinien wystąpić w pewnym punkcie czasu pomiędzy jej wywołaniem a powrotem (punkt linearyzacji).

Zadanie 6. Sprawdź, czy poniższe diagramy reprezentują linearyzowalne historie. Użyj nieformalnej definicji linearyzacji z Definicji 1., a następnie powtórz rozumowanie używając definicji formalnej (ze slajdów).

Definicja 2. Formalną definicję **sekwencyjnej spójności** można streścić w dwóch punktach:

1. Ciąg wywołania metod w programie współbieżnym powinien mieć taki efekt, jak gdyby te metody zostały wykonane w pewnym sekwencyjnym porządku, jedna po drugiej (warunek taki sam, jak w def. linearyzacji).
2. Ciąg wywołania metod w ramach jednego wątku powinien mieć efekt zgodny z ich porządkiem w kodzie tego wątku (porządek programu).

Zadanie 7. Powtórz zadanie 6. tym razem sprawdzając, czy diagramy reprezentują sekwencyjnie spójne historie.

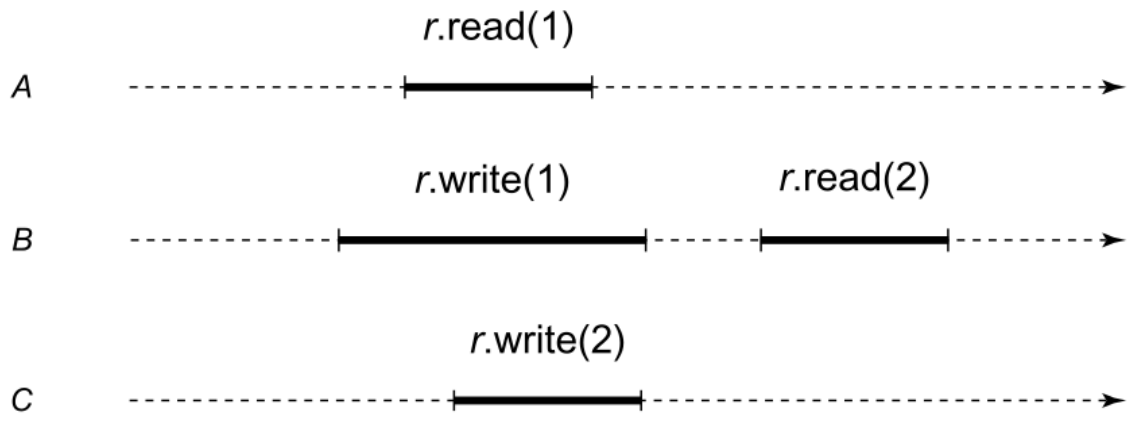

Diagram 1 (zapis/odczyt współdzielonej komórki pamięci r)

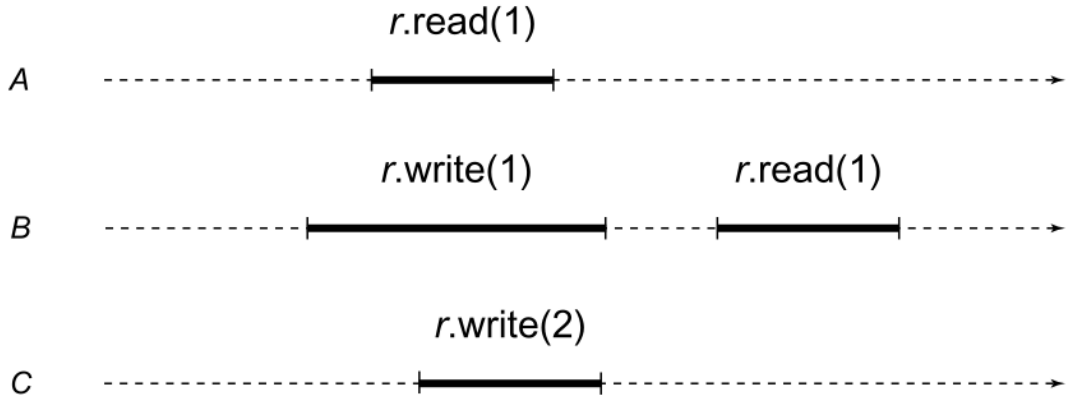

Diagram 2 (zapis/odczyt współdzielonej komórki pamięci r)

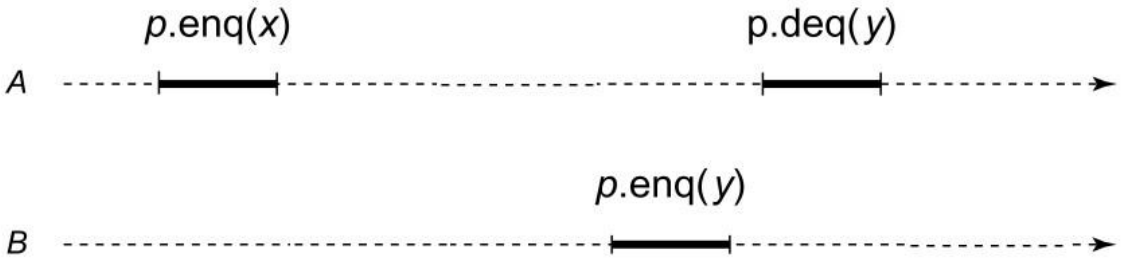

Diagram 3 (p jest kolejką FIFO)

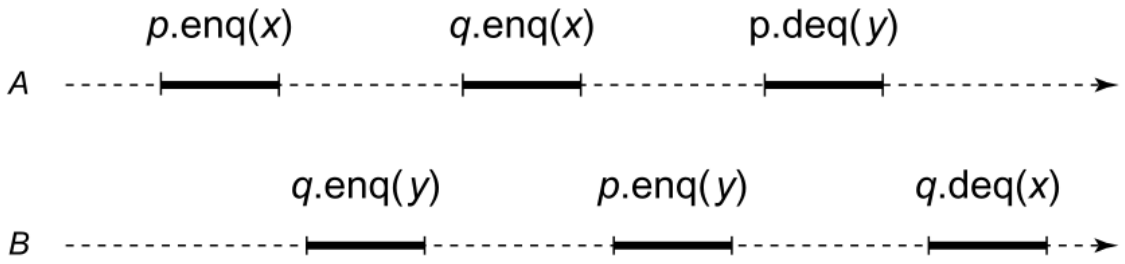

Diagram 4 (p i q są kolejkami FIFO)