Programowanie współbieżne

Lista zadań nr 4

Na ćwiczenia 30 i 31 października 2024

wersja wstępna

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. 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 4. W definicji r-ograniczonego czekania definiujemy sekcję wejściową jako złożoną z *kilku* pierwszych instrukcji algorytmu.

1. Przypomnij dowód tego, że algorytm Petersena jest FCFS (czyli 0-ograniczony).
2. Zmieńmy teraz definicję sekcji wejściowej tak, by składała się po prostu z *pierwszej* instrukcji w algorytmie. Czy, przy zmienionej definicji, algorytm Petersena nadal jest FCFS?
3. Czy istnieje algorytm implementujący zamek (czyli wzajemnie wykluczanie), który jest FCFS przy powyższej definicji sekcji wejściowej? Przeprowadź dowód nie wprost rozważając przypadki, gdy pierwszą instrukcją jest: 1. odczyt tej samej komórki pamięci lub różnych komórek, w zależności od wątku, 2. zapis do różnych komórek, 3. zapis do tej samej komórki.
4. Wyciągnij wniosek, że zmodyfikowana definicja sekcji wejściowej jest *bezsensowna*.

Zadanie 5. Podaj przykład dwóch nietrywialnych diagramów sekwencyjnie spójnych historii oraz dwóch, które nie są sekwencyjnie spójne. Historie mogą dotyczyć dowolnych (czyli wybranych przez Ciebie) współbieżnych struktur danych.

Wskazówka: Możesz wzorować się na diagramach z poprzedniej listy.

Zadanie 6. Pokaż, że sekwencyjna spójność nie ma własności kompozycji.

Wskazówka: slajdy 165-175, PRW-3.pdf. The Art of Multiprocessor Programming 2e, rozdział 3.3.3.

Zadanie 7.

1. Uzasadnij, że klasa **WaitFreeQueue** poprawnie implementuje dwuwątkową kolejkę współbieżną na maszynie z atomowym dostępem do pamięci (dostępy są linearyzowalne). W szczególności pokaż, co dzieje się gdy obydwa wątki współbieżnie wykonują operacje na (prawie) pustej i (prawie) pełnej kolejce.
2. Zauważ, że kolejka ta działa również poprawnie na maszynie z sekwencyjnie spójną pamięcią.

Wskazówka: Kod klasy WaitFreeQueue znajduje się na slajdzie 152. Obserwacja w p. 2. będzie nieomal trywialna.

Zadanie 8. Przypomnij dowód własności wzajemnego wykluczania dla algorytmu Petersona. Pokaż dlaczego ten dowód może się załamać dla procesora o modelu pamięci *słabszym*[1] niż sekwencyjna spójność.

Wskazówka: slajdy 181-183, PRW-3.pdf.

[1] np. dopuszczającym zmianę kolejności wykonywania instrukcji w wątku (ang. *out of order execution*)