Programowanie współbieżne

Lista zadań nr 9

Na ćwiczenia 16. grudnia 2021

Zadanie 1. Zdefiniuj **zasadę lokalności odwołań.** W jaki sposób tę zasadę wykorzystują pamięci podręczne? Dlaczego systemy wieloprocesorowe wymagają zastosowania **protokołów spójności** pamięci podręcznych? Jak działa **protokół MESI**?

Zadanie 2. W jaki sposób mierzy się wydajność implementacji zamków? Wyjaśnij, skąd bierze się różnica w wydajności zamka **TAS** vs. **TTAS** odwołując się do modelu komunikacji w systemach wieloprocesorowych ze spójnymi pamięciami podręcznymi i wspólną szyną danych.

Zadanie 3. Mamy danych n wątków, każdy z nich wykonuje najpierw metodę **foo()** a następnie **bar().** Chcemy zagwarantować, że żaden wątek nie rozpocznie wykonywania **bar()** zanim wszystkie nie skończą wykonywać **foo().** W tym celu pomiędzy wywołaniami **foo()** a **bar()** w kodzie wątków umieścimy **barierę.** Oto dwa pomysły na implementacje bariery:

1. Mamy licznik zabezpieczony zamkiem **TTAS.** Każdy wątek zajmuje zamek, inkrementuje licznik i zwalnia zamek. Następnie aktywnie czeka (wiruje, ang. *spins*) na liczniku oczekując aż osiągnie on wartość n.
2. Mamy n-elementową tablicę wartości boolowskich b[0..n-1], początkowo wypełnioną wartościami false. Protokół bariery składa się z dwóch kroków:
    1. wątek 0 ustawia b[0] na true. Każdy pozostały wątek i (0 < i < n-1) aktywnie czeka na b[i-1] aż ten element osiągnie wartość true, po czym ustawia wartość b[i] na true.
    2. każdy wątek aktywnie czeka aż b[n-1] osiągnie wartość true.

Porównaj wydajność tych dwóch protokołów w systemach wieloprocesorowych ze spójnymi pamięciami podręcznymi i wspólną szyną danych.

Zadanie 4. Poniżej znajduje się alternatywna implementacja zamka CLHLock, w której wątek ponownie wykorzystuje nie węzeł

swojego poprzednika, ale własny. Wyjaśnij, dlaczego ta implementacja jest błędna.

```
public class BadCLHLock implements Lock {
    AtomicReference<Qnode> tail = new AtomicReference<QNode>(new QNode());
    ThreadLocal<Qnode> myNode = new ThreadLocal<QNode> {
        protected QNode initialValue() {
            return new QNode();
        }
    };
    public void lock() {
        Qnode qnode = myNode.get();
        qnode.locked = true; // I'm not done
        // Make me the new tail, and find my predecessor
        Qnode pred = tail.getAndSet(qnode);
        while (pred.locked) {}
    }
    public void unlock() {
        // reuse my node next time
        myNode.get().locked = false;
    }
    static class Qnode { // Queue node inner class
        volatile boolean locked = false;
    }
}
```

Zadanie 5. Opisz działanie zamka CLH z czasem ważności (ang. *timeout*).

Wskazówka: TAoMP 2e, rozdział 7.6.

Zadanie 6. Metoda **isLocked()** wywołana na zamku zwraca wartość true wtedy i tylko wtedy, gdy zamek jest zajęty przez pewien wątek. Podaj implementację metody isLocked() dla następujących zamków: a) TAS, b) CLH, c) MCS.

Zadanie 7. Jakie problemy z wydajnością wystąpią przy zastosowaniu "zwykłych" zamków (np. zamka TAS) w systemach z architekturą NUMA. Wyjaśnij, na przykładzie zamka **HBOLock**, w jaki sposób te problemy rozwiązują zamki hierarchiczne?

Wskazówka: TAoMP 2e, rozdział 7.7 – 7.7.1

Zadanie 8(zadanie bonusowe). Jaka motywacja stoi za ideą zamków kohortowych? Opisz implementację tych zamków, a w szczególności wyjaśnij, do czego służy klasa **TurnArbiter** oraz metoda **alone().**

Wskazówka: TAoMP 2e, rozdział 7.7.2 – 7.7.3