Rozpoczęto	piątek, 10 lutego 2023, 12:15
Stan	Ukończone
Ukończono	piątek, 10 lutego 2023, 14:07
Wykorzystany czas	1 godzina 52 min.
Ocena	Jeszcze nie ocenione

Pytanie **1**

Zakończone

Punkty maks.: 3,00

Co to jest problem ABA?

Problem ABA jest błędem synchronizacji, może się wydarzyć podczas operacji cAS(). Polega na błędnym przekonaniu, że jeśli jakaś wartość jest taka sama jak wcześniej, to nic się nie zmieniło, podczas gdy mogło się zmienić za sprawą innego wątku.

Pytanie **2**

Zakończone

Punkty maks.: 3,00

Rozważmy poznaną przez nas konstrukcję atomowego rejestru MRMW przy pomocy tablicy atomowych rejestrów MRSW. Czy znaczniki czasu (ang. *time-stamps*), które pojawiają się w polach tej tablicy podczas działania rejestru są unikalne? Jeśli tak, wyjaśnij w jaki sposób ta własność jest zachowana. Jeśli nie, wyjaśnij, jaka powinna być wartość zwrócona przez wątek odczytujący ten rejestr, w sytuacji napotkania wielu wpisów o jednakowym, największym znaczniku czasu.

znaczniki czasu w MRMW nie są unikalne, może się zdarzyć, że dwa wątki zapiszą wartość z takim samym timestampem. W przypadku wielu takich samych timestampów decyduje kolejność indeksów

Pytanie **3**

Niepoprawnie

Punkty: 0,00 z 1,00

W modelowym systemie wieloprocesorowym z współdzieloną magistralą:

a. możliwe jest jednoczesne rozgłaszanie, ale nie odbiór

b. jednocześnie wiele procesorów może korzystać z magistrali rozgłaszając i odbierając komunikaty ✖

c. możliwy jest jednoczesny odbiór komunikatów przez wiele procesorów, ale nie jednoczesne rozgłaszanie

Twoja odpowiedź jest niepoprawna.

Poprawna odpowiedź to: możliwy jest jednoczesny odbiór komunikatów przez wiele procesorów, ale nie jednoczesne rozgłaszanie

Pytanie **4**

Poprawnie

Punkty: 1,00 z 1,00

Atomowa migawka nie jest

a. linearyzowalna

b. obiektem o poziomie konsensusu > 1 ✔

c. możliwa do implementacji przy użyciu rejestrów atomowych

Twoja odpowiedź jest poprawna.

Poprawna odpowiedź to: obiektem o poziomie konsensusu > 1

Pytanie **5**

Zakończone

Punkty maks.: 3,00

Odpowiedz krótko na poniższe pytania:

1. Dlaczego sekwencyjnie spójna pamięć nie jest implementowana we współczesnych systemach wieloprocesorowych?

2. Jakie akcje podejmowane przez kompilator i maszynę powodują załamanie się sekwencyjnej spójności programu.

3. Jakie mechanizmy języka Java służą przywróceniu sekwencyjnej spójności kodu.

1. bo jest zbyt mocnym założeniem, bez sekwencyjnej spójności możemy znacznie zoptymalizować wykonywanie kodu
2. zamiana kolejności wykonywania operacji w celu optymalizacji
3. volatile przy deklaracji zmiennych, synchronized przy metodach

Pytanie **6**

Poprawnie

Punkty: 1,00 z 1,00

Rozważmy algorytm licznika oparty na drzewie. Które z następujących faz tego algorytmu nie mogą współbieżnie pracować w różnych poddrzewach tego drzewa:

a. wszystkie fazy mogą pracować współbieżnie w różnych poddrzewach ✔

b. operation i distribution

c. precombine i distribution

d. combine i operation

Twoja odpowiedź jest poprawna.

Poprawna odpowiedź to: wszystkie fazy mogą pracować współbieżnie w różnych poddrzewach

Pytanie **7**

Niepoprawnie

Punkty: 0,00 z 1,00

```
class BarLock implements Lock {
    private int victim;
    public void lock() {
        int i = ThreadID.get();
        victim = i; // let the other go first
        while (victim == i) {} // wait
    }
    public void unlock() {}
}
```

Przedstawiona powyżej implementacja zamka uruchomiona na maszynie z sekwencyjnie spójną pamięcią:

a. spełnia warunek wzajemnego wykluczania, ale dopuszcza zakleszczenie dla dwóch wątków wykonujących lock() współbieżnie

b. spełnia warunek braku zakleszczania, ale nie spełnia warunku wzajemnego wykluczania ✖

c. jest poprawną implementacją zamka dla dwóch wątków

d. spełnia warunek wzajemnego wykluczania, ale dopuszcza zakleszczenie dla wątku pracującego *solo*

Twoja odpowiedź jest niepoprawna.

Poprawna odpowiedź to: spełnia warunek wzajemnego wykluczania, ale dopuszcza zakleszczenie dla wątku pracującego *solo*

Pytanie **8**

Poprawnie

Punkty: 1,00 z 1,00

Które z poniższych stwierdzeń o dowodach za pomocą drzew protokołów jest **nieprawdziwe**:

a. dowód polega na wybraniu dowolnego stanu biwalentnego i pokazaniu, że każda akcja prowadzi z tego stanu do stanu krytycznego ✔

b. są to dowody "nie wprost"

c. wymagają by protokół był wait-free lub lock-free

Twoja odpowiedź jest poprawna.

Poprawna odpowiedź to: dowód polega na wybraniu dowolnego stanu biwalentnego i pokazaniu, że każda akcja prowadzi z tego stanu do stanu krytycznego

Pytanie **9**

Nie udzielono odpowiedzi

Punkty maks.: 3,00

Opisz w skrócie algorytm barier z drzewem turniejowym (ang. *tournament tree barrier*).

Pytanie **10**

Poprawnie

Punkty: 1,00 z 1,00

Wybierz właściwą odpowiedź:

a. Dwuwątkowe kolejki mają poziom konsensusu równy 1, a wielowątkowe są implementowalne przy użyciu wyłącznie rejestrów atomowych.

b. Dwuwątkowe kolejki FIFO mają ten sam poziom konsensusu co kolejki wielowątkowe.

c. Poziom konsensusu wielowątkowych kolejek FIFO wynosi przynajmniej 2. ✔

Twoja odpowiedź jest poprawna.

Poprawna odpowiedź to: Poziom konsensusu wielowątkowych kolejek FIFO wynosi przynajmniej 2.

Pytanie **11**

Niepoprawnie

Punkty: 0,00 z 1,00

```
class ZagLock implements Lock {
    private boolean[] flag = new boolean[2]; // initially false
    private int turn = 1;

    public void lock() {
        int i = ThreadID.get(); // thread-local index, 0 or 1
        int j = 1 - i;
        flag[i] = true;

        while (flag[j]) {
            if (turn != i) {
                flag[i] = false;
                while (turn != i) { }
                flag[i] = true;
            }
       }
    }

    public void unlock() {
        int i = ThreadID.get();
        turn = 1 - i;
        flag[i] = false;
    }
}
```

Przedstawiona powyżej implementacja zamka uruchomiona na maszynie z sekwencyjnie spójną pamięcią:

- a. spełnia warunek wzajemnego wykluczania, ale dopuszcza zakleszczenie
- b. nie dopuszcza zakleszczeń, ale nie spełnia warunku wzajemnego wykluczania
- c. każda z pozostałych odpowiedzi jest fałszywa ✖
- d. spełnia warunek wzajemnego wykluczania oraz niegłodzenia (ang. starvation-free), a zatem jest poprawną implementacją zamka dla dwóch wątków

Twoja odpowiedź jest niepoprawna.

Poprawna odpowiedź to: spełnia warunek wzajemnego wykluczania oraz niegłodzenia (ang. starvation-free), a zatem jest poprawną implementacją zamka dla dwóch wątków

Pytanie **12**

Niepoprawnie

Punkty: 0,00 z 1,00

Które z następujących zdań o zamku kolejkowym Andersona (ang. Anderson queue lock, ALock) jest prawdziwe:

a. Wszystkie pozostałe odpowiedzi są fałszywe.

b. Program używa pojedynczego zamka Andersona, którego tablica flag ma N pozycji. Wątków w programie jest M. Zamek zadziała poprawnie wtedy i tylko wtedy gdy M <= N.

c. W implementacji, w której pozbyliśmy się fałszywego współdzielenia (ang. *false-sharing*) za pomocą wypełnień (ang. *padding*), o dostęp do każdej komórki pamięci zamka rywalizują co najwyżej dwa wątki. ✖

d. Każdy wątek ma przypisany indeks w tablicy flag, niezmienny podczas działania programu.

Twoja odpowiedź jest niepoprawna.

Poprawna odpowiedź to: Wszystkie pozostałe odpowiedzi są fałszywe.

Pytanie **13**

Zakończone

Punkty maks.: 3,00

Wyjaśnij, z czego wynika różnica w wydajności pomiędzy licznikami implementowanymi za pomocą pojedyńczego rejestru RMW z operacją getAndIncrement(), a licznikami rozproszonymi, implementowanymi za pomocą drzew lub sieci równoważników.

Wynika to z braku potrzeby zajmowania i zwalniania zamka. Liczniki rozproszone pozwalają na niezależną pracę wątków. W sieciach róznoważników podoperacje działają niezależnie, co redukuje czas do logarytmicznego

Pytanie **14**

Zakończone

Punkty maks.: 3,00

Do czego służy klasa AtomicMarkableReference<> i jakie metody ona oferuje? Wymień przynajmniej 2 przykłady algorytmów, w których implementacji w języku Java wykorzystuje się tę klasę. Czy można zamiast powyższej klasy wykorzystać tam AtomicReference<>?

AtomicMarkableReference<> opakowuje referencję, dodając do niej pole typu boolean. Oferuje metody get(), isMarked(), getReference(), compareAndSet(). Przykłady używające atomicMarkableRefernce to LockFreeList, Window. Nie można wykorzystać AtomicReference, bo tracimy pole marked

Pytanie **15**

Zakończone

Punkty maks.: 3,00

Oto standardowa implementacja kolejki przeznaczonej do wykorzystania w dwuwątkowym systemie z sekwencyjnie spójną pamięcią:

```
class WaitFreeQueue<T> {
    int head = 0, tail = 0;
    T[] items;
    public WaitFreeQueue(int capacity) {
        items = (T[]) new Object[capacity];
    }
    public void enq(T x) throws FullException {
        if (tail - head == items.length)
            throw new FullException();
        items[tail % items.length] = x;
        tail++;
    }
    public T deq() throws EmptyException {
        if (tail - head == 0)
            throw new EmptyException();
        T x = items[head % items.length];
        head++;
        return x;
    }
}
```

Opisz skrótowo wszystkie błędy w działaniu tej kolejki, które mogą nastąpić gdy umożliwimy dwóm wątkom współbieżne wywoływanie enq().

Załóżmy, że wątki A i B wykonują enq()

A wykonuje enq(x), zapisuje x ale zostaje wywłaszczony i nie zdąży zwiekszyć tail

B wykonuje enq(y) i zapisuje do tego samego miejsca gdzie zapisał A, element x zostaje zapomniany

Problem może wystąpić również wtedy, gdy w kolejce jest tylko jedno miejsce, wystąpi sytuacja opisana powyżej i nie zostanie wtedy wyrzucony wyjątek FullException(), ponieważ jest on sprawdzany na poczatku, a dopiero jak wątek A się wybudzi to rozmiar kolejki będzie wiekszy niż dopuszczalny

Pytanie **16**

Nie udzielono odpowiedzi

Punkty maks.: 3,00

Odpowiedz na poniższe pytania dotyczące współbieżnej implementacji tablicy haszującej opartej na rekurencyjnym porządku dzielonym.

1. Jaka jest rola strażników, występujących na liście elementów przechowywanych w takiej tablicy haszującej? Czy ich wprowadzenie było tylko optymalizacją, czy też rozwiązują one jakiś istotny problem?
2. Jeśli rozszerzanie tablicy wykonywane jest inkrementacyjnie, to w jaki sposób wyznaczyć pozycję na liście, na której powinien znaleźć się strażnik nowopowstałego kubełka?
3. Jaka jest złożoność obliczeniowa zadania z poprzedniego podpunktu?

Pytanie **17**

Zakończone

Punkty maks.: 3,00

Rozważmy metodę add(e) klasy OptimisticList (implementacja listy współbieżnej stosująca synchronizację optymistyczną). Po znalezieniu sąsiadujących elementów pred oraz curr, pomiędzy które należy wstawić element e i założeniu na nich zamków, metoda weryfikuje m.in. osiągalność elementu pred z głowy listy. Podaj przykład błędnego wykonania metody add(e), jeśli pominiemy taką weryfikację.

podczas przechodzenia nie wykonujemy locków, więc gdy znajdujemy pred i curr to inny wątek może nam zmodyfikować listę zanim my wykonamy tam locka (np dodając coś pomiędzy pred i curr), wtedy jak my byśmy dodali coś pomiędzy pred i curr to wykonamy dalej node.next = curr i pred.next = node i tym samym wyrzucilibyśmy z listy element, który dodał tam inny wątek tuż przed tym, jak my wykonaliśmy locka na pred i curr

Pytanie **18**

Zakończone

Punkty maks.: 3,00

Rozważamy poznaną przez nas konstrukcję m-wartościowego regularnego rejestru MRSW używającą tablicy binarnych regularnych rejestrów MRSW. W jaki sposób w tej tablicy pamiętamy wartości z przedziału [0,..m-1]? Czy podczas zapisów i odczytów do tak skonstruowanego rejestru w tablicy może pojawić się wiele wartości niezerowych?

mamy tablice bitów wielkości m, zapis polega na zapaleniu konkretnego bitu i wyzerowaniu od góry do dołu mniejszych bitów, read idzie do góry i zwraca liczbę będącą pierwszym indeksem z zapalonym bitem, może tam być wiele wartości niezerowych, jednak to nie przeszkadza, ponieważ read zwróci tę najmniejszą, warunek regularności jest spełniony

Pytanie **19**

Zakończone

Punkty maks.: 3,00

Podaj najważniejszy argument uzasadniający, dlaczego linearyzacja, a nie sekwencyjna spójność, jest właściwą notacją dla poprawności algorytmów współbieżnych.

Linearyzacja pozwala na przedstawienie wydarzeń na jednej, spójnej osi czasu. Pomaga to w naturalnym zrozumieniu działania algorytmu, a sekwencyjna spójność gwarantuje nam jedynie kolejność w obrębie jednego wątku

Pytanie **20**

Zakończone

Punkty maks.: 3,00

Wyjaśnij, z czego wynika różnica w wydajności pomiędzy współbieżnym niewstrzymywanym (ang. *lock-free*) stosem (klasa LockFreeStack), a tym samym stosem używającym dodatkowo tablicy eliminacji (klasa EliminationBackoffStack).

Różnica wynika z niewykonywania 'bezsensownych' operacji dodawania i zdejmowania ze stosu w przypadku, w którym dużo wątków jednocześnie próbuje dodać i zdjąć elementy. Z tablicą eliminacji wątki zdejmujące ze stosu w ogóle nie będą go modyfikowały, tylko 'odbiorą' szukaną wartość od wątków próbujacych dodać na stos. Pozwala to na istotne zwiększenie wydajności w przypadku działania wielu wątków w sposób zbalansowany (kiedy około tyle samo średnio 'dodaje' co 'zabiera')

◄ Tematy referatów

Przejdź do...