

Programowanie współbieżne

Lista zadań nr 11

Na ćwiczenia 18 grudnia i 7 stycznia

Zadanie 1. Rozważmy standardową sekwencyjną (jednowątkową) implementację listową zbioru (elementy przechowywane na liście uporządkowanej względem kluczy, bez powtórzeń, ze strażnikami). Dlaczego zastąpienie wszystkich operacji przypisania referencji w funkcjach `add()` oraz `remove()` operacjami `compareAndSet()` nie daje w wyniku poprawnej współbieżnej implementacji zbioru? W jaki sposób użycie pola `marked` oraz klasy `AtomicMarkableReference<T>` pomaga w rozwiązaniu powstałego problemu?

Wskazówka: TAoMP2e, Fig. 9.21

Zadanie 2. Opisz w dokładny sposób działanie metody `find()` z klasy `Window` oraz metod `add()`, `remove()` i `contains()` z klasy `LockFreeList`. W szczególności, dla każdego wywołania `compareAndSet()` występującego w treści tych metod wymień wszystkie powody, dla których może ono zawieść (zwrócić `false`). Dlaczego rezultat drugiego wywołania `compareAndSet()` w metodzie `remove()` nie jest sprawdzany? Czy można je usunąć nie tracąc poprawności implementacji?

Wskazówka: TAoMP2e, r. 9.8

Zadanie 3. Założymy, że w metodzie `add()` klasy `LockFreeList` okazało się, że niezbędny jest kolejny obrót pętli `while(true)`, ponieważ `pred` nie wskazuje już na `curr`, ale `pred` nie ma ustawionego pola `marked`. Czy w tej sytuacji koniecznie musimy przeglądać całą listę od początku?

Zadanie 4. Uzasadnij, że metody `add()` i `remove()` klasy `LockFreeList` są **niewstrzymywane** (ang. *lock-free*), a metoda `contains()` jest **nieczekająca** (ang. *wait-free*).

Zadanie 5. Uzasadnij, że metody `add()`, `remove()` oraz `contains()` klasy `LazyList` są linearyzowalne. Czy dla każdej z

tych metod jesteś w stanie wskazać konkretny punkt linearyzacji w jej kodzie?

Zadanie 6. Przypomnij, jak działa wspólnie kolejka ograniczonego rozmiaru **BoundedQueue**. W szczególności, w jaki sposób działają i jak są wykorzystywane zmienne warunkowe **notEmptyCondition** i **notFullCondition**. Czy istnieje taka sekwencja wykonania metod **enq()** i **deq()**, że zmienna **size** staje się ujemna?

Wskazówka: TAoMP2e, r. 10.3 – klasa BoundedQueue, r. 8.2 – monitory i zmienne warunkowe