Programowanie wielowątkowe w Javie

Łukasz Andrzejewski

2 Czym jest wątek?

- przepływ sekwencji sterowania w programie, posiadający własny stos wywołań
- wątki w ramach procesu mają dostęp do wspólnej przestrzeni adresowej
- aplikacja rozpoczyna działanie w wątku głównym (ang. main thread), który może tworzyć i uruchomiać kolejne wątki
- wątki konkurują o dostęp do fizycznych zasobów komputera, dając możliwość i/lub iluzję pracy równoległej

3 Tworzenie wątku

- w celu uruchomienia nowego wątku należy:
  - zaimplementować interfejs Runnable - instrukcje umieszczone w metodzie run() będą wykonywane w czasie pracy wątku
  - stworzyć instancję wątku (klasa Thread) i zainicjajizować ją obiektem typu Runnable
  - uruchomić wątek przy użyciu metody start()

4 Wykonywanie wątków

Programowanie wielowątkowe w Javie - wybrane zagadnienia

- model wywłaszczeniowy zakłada, że wątek jest w stanie Running dopóki mechanizm szeregowania wątków nie zdecyduje o tym, że inny wątek powinien uzyskać dostęp do zasobów
- Java udostępnia metody statyczne pozwalające na sterowanie wykonaniem wątków:
  - yield() - aktualny wątek oddaje dostęp do procesora, Scheduler wybiera wątek z puli Runnable i umożliwia mu działanie
  - sleep() – uśpienie aktualnego wątku na określony czas

5 Metody sterujące wątkami

Programowanie wielowątkowe w Javie - wybrane zagadnienia

- join() – powoduje zatrzymanie wykonania bieżącego wątku do momentu zakończenia pracy innego wątku lub upływu określonego czasu
- setPriority() – pozwala na ustalenie priorytetu wątku
- setDeamon() – powoduje przejście wątku do stanu demonicznego
- isAlive() – zwraca informację czy wykonano metodę start() i czy wątek nie zakończył jeszcze działania
- stop() – natychmiastowe zatrzymanie pracy wątku - niezalecane
- interrupt() – bezpieczne zatrzymanie wątku, oparte o ustawienie flagi

6 Ochrona współdzielonych danych

- w celu ochrony integralności danych można wykorzystać wbudowany mechanizm monitora do obiektu
- monitor to znacznik, który w danej chwili może być przejęty tylko przez jeden wątek (poprzez realizację bloku lub metody synchronizowanej)
- jeśli wątek posiadający monitor zostanie wywłaszczony nie oddaje monitora
- należy ograniczać liczbę instrukcji synchronizowanych - nie są one wykonywane współbieżnie, dlatego mogą stanowić "wąskie gardło" aplikacji. Dodatkowo powstają narzuty związane z przejęciem/zwrotem monitora

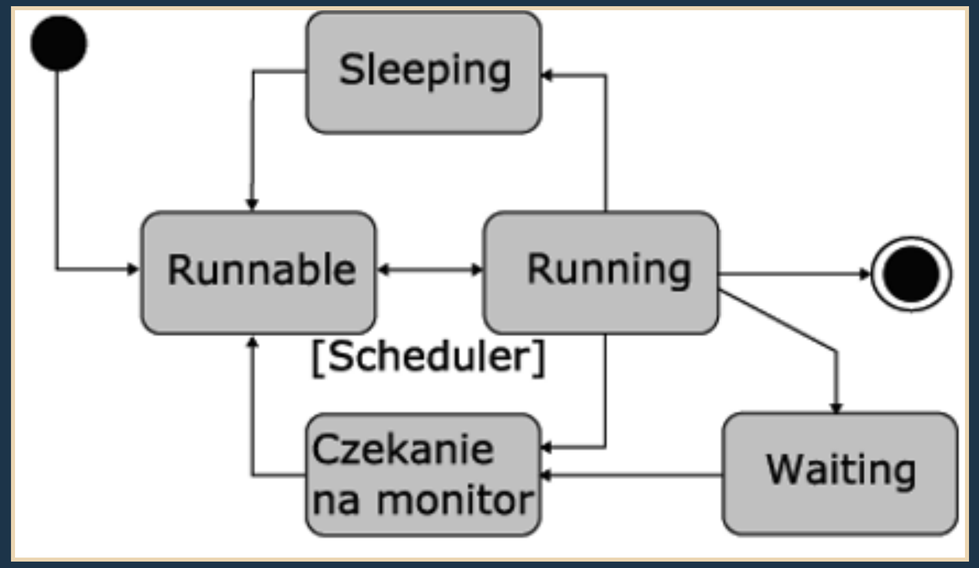
7 Zakleszczenie

- występuje gdy wątek posiadający monitor do obiektu oczekuje dostępu do obiektu, którego monitor jest zajęty przez inny wątek, który oczekuje na dostęp do monitora posiadanego przez wątek pierwszy
- sytuacja zakleszczenia często jest trudna do wykrycia (zależności czasowe)

8

Koordynacja wątków

Programowanie wielowątkowe w Javie - wybrane zagadnienia



9 Internal Java Memory Model

Programowanie wielowątkowe w Javie - wybrane zagadnienia

- zmienne na poziomie metod (typy prymitywne i referencje) są przechowywane na stosie i kopiowane per wątek
- obiekty zapisywane są na stercie i mogą być współdzielone między wątkami, zmienne instancyjne i statyczne przechowywane są na stercie wraz z obiektem (nawet prymitywne) i definicją klasy

Klasyczny, sprzętowy model pamięci

- komputery mogą posiadać więcej niż jeden procesor, z których każdy może mieć wiele rdzeni, co umożliwia równoczesne działanie wielu wątków
- każdy procesor zawiera zbiór rejestrów oraz pamięć podręczną (często wielopoziomową)
- operacje na rejestrach są dużo bardziej wydajne niż operacje na pamięci głównej (RAM)
- operacje na pamięci podręcznej CPU są wolniejsze niż na rejestrach, ale szybsze niż na pamięci głównej (RAM)
- dostęp do danych w RAM odbywa się najczęściej pośrednio (przez pamięć cache i rejestry)

11 Sprzętowy model pamięci vs. Java Memory Model

- sprzętowy model pamięci nie rozróżnia stosu i sterty (oba obszary są częścią RAM)
- fragmenty stosu i sterty mogą być przechowywane na poziomie rejestrów i pamięci podręcznej CPU
- przechowywanie zmiennych i obiektów w różnych obszarach pamięci może prowadzić do problemów
  - widoczność zmian stanu współdzielonych zmiennych
  - race condition przy odczycie, wykorzystaniu i zapisie współdzielonych zmiennych

12 Słowo kluczowe volatile

- jeśli jeden wątek zmienia wartość zmiennej balance, a drugi dokonuje jej odczytu, volatile gwarantuje spójność (widoczność zmian)

```
public class Account {

   volatile int balance = 0;

}
```

Słowo kluczowe volatile vs. synchronizacja

- volatile może być stosowane na poziomie zmiennych prymitywnych, synchronizacja (synchronized, typy atomowe) może odnosić się jedynie do typów referencyjnych
- volatile nie jest związane z blokadą (zamkiem), w związku z tym nie nadaje się do operacji typu atomic read-update-write

14 Spójność na poziomie aplikacji wielowątkowych

- Mutual Exclusion - tylko jeden wątek może wykonywać sekcję krytyczną
- Visibility - zmiany realizowane przez jeden wątek są widoczne dla innych wątków
- synchronizacja z użyciem blokad spełnia oba warunki kosztem wydajności
- volatile - spełnia tylko drugi warunek, koszty wydajnościowe są mniejsze ale nadal występują (praca na „pamięci głównej”, brak wybranych optymalizacji)

Priorytet wątku

- menadżer wątków/scheduler w Javie przydziela zasoby procesora bazując na priorytecie wątku stosując jednocześnie time slicing
- każdy wątek w Javie posiada priorytet w zakresie 1-10 (standardowo 5)
- każdy nowo utworzony wątek dziedziczy priorytet po wątku który go stworzył
- priorytet wątku może zostać zmieniony z poziomu kodu w dowolnym momencie działania programu
- priorytety wątków w Javie są mapowane na priorytety w systemie operacyjnym (ich ilość może być różna w zależności od systemu)

Kolekcje synchronizowane

- większość kolekcji w języku Java nie gwarantuje bezpieczeństwa bez użycia zewnętrznej synchronizacji
- w przypadku prostych/pojedynczych operacji kolekcje można opakować stosując statyczne metody z Collections np. synchronizedList(), synchronizedMap(). W rezultacie dostęp do metod kolekcji jest synchronizowany (obiektem synchronizacji jest instancja kolekcji)
- w przypadku operacji złożonych takich jak jednoczesny odczyt i zapis należy użyć innych mechanizmów np. blokady jawne lub niejawne

17 Kolekcje współbieżne

Programowanie wielowątkowe w Javie - wybrane zagadnienia

- synchronizacja kolekcji może znacząco wpływać na wydajność realizowanych operacji (zwłaszcza jeśli są one złożone np. iteracja po wszystkich elementach)
- kolekcje współbieżne są bezpieczne wielowątkowo, ale jednocześnie nie ograniczają możliwości wykonywania operacji przez wiele wątków np.
  - copy on write collections np. CopyOnWriteArrayList, CopyOnWriteArraySet
  - compare and swap (CAS) np. ConcurrentLinkedQueue, ConcurrentSkipListMap
  - oparte o blokady jawne np. ConcurrentHashMap, BlockingQueue
- powyższe kolekcje nie rzucają wyjątkiem ConcurrentModificationException

Wielowątkowość na poziomie GUI (Swing)

- operacje związane obsługą UI powinny realizowane się w specjalnym wątku (event dispatch thread - EDT), który gwarantuje bezpieczeństwo w środowisku wielowątkowym
- operacje związane z logiką biznesową, przetwarzaniem powinny być realizowane w tle (worker threads), zwykle z użyciem SwingWorker