Politechnika Wrocławska

Platformy Programistyczne
Programowanie współbieżne

Agata Migalska

3/4 czerwca 2014

Wątki

1. Wątki
   - Drzemka
   - Przerwania
2. Synchronizacja wątków

Drzemka

`Thread.sleep`

- powoduje, że aktualny wątek jest zawieszony na określony czas,
- pozwala innym wątkom aplikacji lub innym aplikacjom zająć czas procesora,
- pozwala na spowolnienie działania programu,
- pozwala na oczekiwanie, aż inny wątek, który ma wymagania czasowe, skończy swoje zadanie.

Przerwania

Przerwania

- Przerwanie to sugestia dla wątku, że powinien przestać robić to co robi, i dla odmiany zrobić coś innego. Programista decyduje, co dokładnie powinno się stać - zazwyczaj taki wątek kończy działanie.

Złączenia

Złączenia wątków czyli `join`

```
t.join();
```

Metoda `join` pozwala jednemu wątkowi poczekać na zakończenie działania innego wątku.

Złączenia

Złączenia wątków czyli join

```
t.join();
```

Metoda join pozwala jednemu wątkowi poczekać na zakończenie działania innego wątku.

Przykład - Kto na kogo czeka?

```
Thread t = new Thread(new Zadanie());
t.start();
t.join();
```

Wątek główny, który stworzył wątek t, czeka na jego zakończenie.

Synchronizacja wątków

1 Wątki

2 Synchronizacja wątków

Problemy ze śluzami powietrznymi

Na nowo budowanej stacji orbitalnej odbywa się posiedzenie sztabu programistów poświęcone analizie projektu kontroli systemu śluz powietrznych. Wraz ze zbliżającym się otwarciem stacji, planowana ilość wyjść w przestrzeń kosmiczną miała drastycznie wzrosnąć, a ruch w śluzach powietrznych - zarówno do wnętrza jak i na zewnątrz statku - miał być bardzo wysoki.

"Każda śluza powietrzna wyposażona jest we wzmocnione terminale graficzne. Za każdym razem, gdy ktoś będzie opuszczał stację lub wchodził do niej, będzie musiał skorzystać z tych terminali, żeby zapoczątkować sekwencje użycia śluzy" - powiedział główny programista.

Problemy ze śluzami powietrznymi c.d.

Zaproponowana sekwencja metod dla opuszczenia stacji:

1. weryfikujStatusPortalu();
2. wyrównajCiśnienieWŚluzie();
3. otwórzWewnętrzneDrzwiŚluzy();
4. potwierdźObecnośćPasażera();
5. zamknijWewnętrzneDrzwiŚluzy();
6. dekompresujŚluzę();
7. otwórzZewnętrzneDrzwiŚluzy();
8. potwierdźOpróżnienieŚluzy();
9. zamknijZewnętrzneDrzwiŚluzy();

Problemy ze śluzami powietrznymi c.d.

To na czym w zasadzie problem polega?

A co się stanie jeżeli w trakcie opuszczania stacji przez jednego pasażera (wątku), inny pasażer (wątek) zainicjuje sekwencję wyjścia lub wejścia?

Synchronizacja

`synchronized`

- Tylko jeden wątek ma jednocześnie dostęp do metody lub bloku.
- Synchronizacja zabezpiecza przez odczytaniem niespójnego stanu obiektu.
- Wątek wchodzący do bloku lub metody synchronizowanej "widzi" efekty wszystkich poprzednich modyfikacji chronionych przez tą samą blokadę.

Synchronizowane metody

```
public class Counter{
    private static int count = 0;
    public static synchronized int getCount(){
        return count;
    }
    public synchoronized setCount(int count){
        this.count = count;
    }
}
```

Synchronizowane bloki

Przykład: Blokada z podwójnym zatwierdzeniem (*ang. double checked locking*)

```
public class Singleton{
    private static volatile Singleton INSTANCE;
    public static Singleton getInstance(){
      if(INSTANCE == null){
          synchronized(Singleton.class){
              if(INSTANCE == null) {
                  INSTANCE = new Singleton();
              }
          }
      }
      return INSTANCE;
}
```

volatile

```
private static volatile boolean stopRequested;
```

Modyfikator volatile

- Gwarantuje, że każdy wątek odczytujący wartość zmiennej będzie "widział" ostatnio zapisaną wartość.
- Jest wystarczający, jeżeli wszystkie operacje wykonywane na zmiennej są atomowe.
- Typowy błąd - Operacja x++ nie jest operacją atomową - jest to odczyt, modyfikacja zmiennej i zapis nowej wartości.

Dlaczego nie synchronizować wszystkiego

Zasada minimum

Wewnątrz bloku synchronizowanego wykonuje się możliwie niewiele operacji.

- założenie blokady
- odczytanie współdzielonych danych
- modyfikacja danych
- zwolnienie blokady

Dlaczego nie synchronizować wszystkiego

Koszty synchronizacji

- stracona możliwość zapewnienia współbieżności,
- opóźnienia powodowane koniecznością upewnienia się, że każdy rdzeń procesora ma spójny obraz zawartości pamięci,
- brak możliwości optymalizacji wykonywanego kodu przez maszynę wirtualną.