

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών Τμήμα Πληροφορικής & Τηλεπικοινωνιών

Προηγμένες Μέθοδοι Προγραμματισμού

ΠΜΣ 2022-23 (M135.CS1E, M135.CS23B, M135.IC1E, παλαιό: M117)

Concurrent programming (2)

Δρ. Κώστας Σαΐδης (saiko@di.uoa.gr)

Liveness

- The ability of the multi-threaded, concurrent application to execute in a timely manner.
- Three liveness problems: deadlock, starvation, livelock.

Deadlock

- A situation where two or more threads are waiting for each other, blocking indefinetely.
- There are four conditions that must hold simultaneously for a deadlock to occur:
 - Mutual exclusion: there is at least a non-shareable resource.
 - Hold and wait: a requesting thread holds a resource and waits for some other requested resource.
 - No-preemption allowed: a resource held by the lock cannot be ``taken back''.
 - Circular wait: threads form a chain where each one waits for a resource held by the next.

Example 08 (BankAccount & BankTransfer)

How to avoid deadlocks

- Provide an ordering to your locks.
- Don't synchronize randomly.

Example 08 (contd.)

Starvation

- A thread is unable to gain access to shared resources and, thus, is unable to make progress.
- This happens when shared resources are made unavailable (locked) for long periods of time.

Livelock

- In cases where a thread (A) act in response to the actions of another thread (B) and the latter also responds to actions of some other thread (C), may lead to livelocked threads.
- As with deadlock, livelocked threads are unable to make progress; the threads are not blocked — they are simply too busy responding to each other to resume any actual work.

Guarded blocks, thread signaling and coordination

 The most common thread coordination idiom is the guarded block; a block begins by polling a condition that must be true before the block can proceed.

Example

```
public synchronized void guardedAction() {
    while(!safe) {
        try {
            wait();
            }
        catch(InterruptedException ie) {}
        ...
        // do the action
    }
}
```

The builtin Object'swait()andnotify()methods

- These methods effectively allow threads to signal each other.
- They are available to all Java objects (they reside in the java.lang.Object class).
- A call to obj.wait() suspends the current thread. The thread is said to be "waiting on" the obj.
- Another thread calls obj.notify() Or obj.notifyAll().
 This "wakes up" the threads waiting on the obj.
- You need to have the lock/monitor of an object to be able to call wait/notify.

The wait() and wait(long) methods

- These methods, when called on an object, cause the current thread to wait until either another thread invokes notify() / notifyAll() on this object, or a specified amount of time has elapsed (timed waiting).
- The current thread releases the object's lock/monitor during a wait.
- When a thread is interrupted during a wait, the thread wakes up and the wait() method returns by throwing an InterruptedException.

The notify() and notifyAll() methods

- The first wakes up a single thread that is waiting on this object's monitor. If many threads are waiting on this object, one of them (arbitrary) is chosen to be awakened.
- The second wakes up all threads that are waiting on this object's monitor.

The current thread (the invoker of notify() or notifyAll()) should release the object's lock in order for the awakened threads to compete again to obtain the object's lock.

When to use guarded blocks with wait/notify signaling?

- In various scenarios that require thread communication
- Producer-Consumer (classic example):
 - A data structure,
 - where one or more threads write to it
 - and one or more thread read from it.

Example 09 (A logging producer/consumer)

- Multiple threads (producers) use a common logger to log application events.
- The logger writes events to standard output (consumer).

Can we avoid synchronization?

- Yes, when we don't share state!
- Making our objects immutable guarantees that no synchronization will be required (the state of our objects does not change).
- Immutability: since the state of immutable objects cannot change, the objects cannot be corrupted by thread interference or observed in an inconsistent state.