CS601: Principles of Software Development

Thread Liveness.

Olga A. Karpenko

This presentation is based on the materials of Prof. Engle.

Motivation

- We want threads to execute in a timely manner
- Liveness problems
  - Threads can die prematurely (*deadlock*)
  - Threads can starve & take a long time (*starvation*)
  - Threads are busy responding to each other, but not able to do any work (*livelock*)

Deadlock

- Two or more threads wait for each other to finish work
- Threads are indefinitely blocked and never complete
  - The threads are effectively dead
  - Similar effect as an infinite loop

Deadlock Example

```
public void transfer(Account a, Account b, int amount)
{
   lock(a);
   lock(b);
   withdraw(b, amount);
   deposit(a, amount);
   unlock(b);
   unlock(a);
}
```

Deadlock Example

```
public void transfer(Account a, Account b, int amount)
{
   lock(a);
   lock(b);
   withdraw(b, amount);
   deposit(a, amount);
   unlock(b);
   unlock(a);
}
```

What happens if we do (concurrently):

- Thread 1: transfer(John, Alice, 100);
- Thread 2: transfer(Alice, John, 200);

Deadlock Example

Thread 1	Thread 2
lock(John's acct)	**lock(Alice's acct)**
lock(Alice's acct)	lock(John's acct)
withdrawal and deposit	withdrawal and deposit
unlock(Alice's acct)	unlock(John's acct)
unlock(John's acct)	unlock(Alice's acct)

- Thread 1 can not proceed, needs Alice's lock, held by Thread 2
- Thread 2 can not proceed, needs John's lock, held by Thread 1

Example

- Deadlock.java

Deadlock Avoidance

- Avoid obtaining multiple locks if possible
- Try to obtain locks in the same order
- Avoid dependencies and cycles
  - task 1 depends on task 2 depends on task 1

Starvation

- Lower priority threads are starved of the resource
  - Take too long to complete or
  - Never complete
- A higher priority thread prevents a lower priority thread from accessing a resource
  - Resource may be CPU time or something else
  - Often caused by overzealous synchronization

http://docs.oracle.com/javase/tutorial/essential/concurrency/starvelive.html

Livelock

- A thread triggers another thread,
- Which triggers the previous thread,
- And so on…!

Livelock

- Threads spend all effort on responding to each other
- Not blocking each other, so still "lively"
- Not able to perform any *real* work

Livelock

- See `Livelock.java`