CS601: Principles of Software Development

Thread Liveness.

Olga A. Karpenko

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

Motivation

- We want healthy threads
  - Threads should execute in a timely manner
- Liveness problems
  - Threads can die prematurely (*deadlock*)
  - Threads can starve & take a long time (*starvation*)
  - Threads can be too distracted (*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

#	transfer(a, b, amount)	transfer(b, a, amount)
1	`lock(a);`	`lock(b);`
2	`lock(b);`	`lock(a);`
3	`withdraw(b, amount);`	`withdraw(a, amount);`
4	`deposit(a, amount);`	`deposit(a, amount);`
5	`unlock(b);`	`unlock(a);`
6	`unlock(a);`	`unlock(b);`
7	*Will this finish?*	

Deadlock Example

-

#	transfer(a, b, amount)	transfer(b, a, amount)
1	lock(a);	lock(b);
2	lock(b); // must wait	lock(a); // must wait
3	~~withdraw(b, amount);~~	~~withdraw(a, amount);~~
4	~~deposit(a, amount);~~	~~deposit(a, amount);~~
5	~~unlock(b);~~	~~unlock(a);~~
6	~~unlock(a);~~	~~unlock(b);~~
7	**DEADLOCK on Line 2!**	

Example

- Deadlock.java

Deadlock Avoidance

- Hard to detect and predict
- 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…

- Threads spend all effort on responding to each other
  - Not blocking each other, so still "lively"
  - Locked in a loop preventing progress
  - Sometimes caused by deadlock prevention!