USF UNIVERSITY OF SAN FRANCISCO

CHANGE THE WORLD FROM HERE

Thread Liveness

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science

Motivation

- We want healthy threads (i.e. **thread liveness**)
  - *Thread should execute in a timely manner*
- Several situations to avoid (i.e. **liveness problems**)
  - Threads can die prematurely (*deadlock*)
  - Threads can starve and take a long time (*starvation*)
  - Threads can be too distracted (*livelock*)

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

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Deadlock

- Occurs when two or more threads must wait for each other to finish work
- Threads are indefinitely blocked and never complete
  - The threads are effectively dead (hence deadlock)
  - Similar effect as an infinite loop

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

Deadlock Example

```
1  public void transfer(Account a, Account b, int amount) {
2      lock(a);
3      lock(b);

4      withdraw(b, amount);
5      deposit(a, amount);

6      unlock(b);
7      unlock(a);
1  }
```


Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

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?*	

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

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!**	

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Deadlock Avoidance

- **Detection** and **prevention** difficult
  - Must turn to heuristics for **avoidance**
- Avoid obtaining multiple locks if possible
- Try to obtain locks in same order
- Avoid dependencies and cycles

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Starvation

- Occurs when 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
- Lower priority threads are starved of the resource, and take too long (or never) complete

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

Livelock

- Occurs when a thread triggers another thread, which triggers the previous thread, and so on
- Threads spend all effort on responding to each other
  - Threads are not blocking each other, so still "lively" but locked in a loop preventing progress
  - *Sometimes caused by deadlock prevention!*

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

CHANGE THE WORLD FROM HERE

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science