Lecture 10: Deadlock

10.0 Main Points:

Definition of deadlock

Conditions for its occurence

Solutions for breaking and avoiding deadlock

10.1 Definitions

10.1.1 Resources

Threads -- active

Resources -- passive, things needed by thread to do its job (CPU, disk space, memory, locks)

Two kinds of resources:

Preemptable -- can take it away (CPU)

Non-preemptable -- must leave with thread

10.1.2 Starvation vs. Deadlock

Starvation -- thread waits indefinitely

Deadlock -- circular waiting for resources

Deadlock implies starvation, but not vice versa

Example:

Thread A	Thread B
`x.P();` `y.P();`	`y.P();` `x.P();`

10.2 Conditions for deadlock

10.2.1 Motivation

a. Deadlock can happen with any kind of resource.

b. Deadlocks can occur with multiple resources.

For example: one thread grabs the memory it needs
another grabs disk space
another grabs the tape drive

each waits for the other to release.

c. Deadlock can occur whenever there is waiting.

Example: dining lawyers

Each lawyer needs two chopsticks to eat. Each grabs chopstick on the right first.

What if all grab at the same time? Deadlock.

10.2.2 Conditions

Conditions for deadlock -- without **all** of these, can't have deadlock:

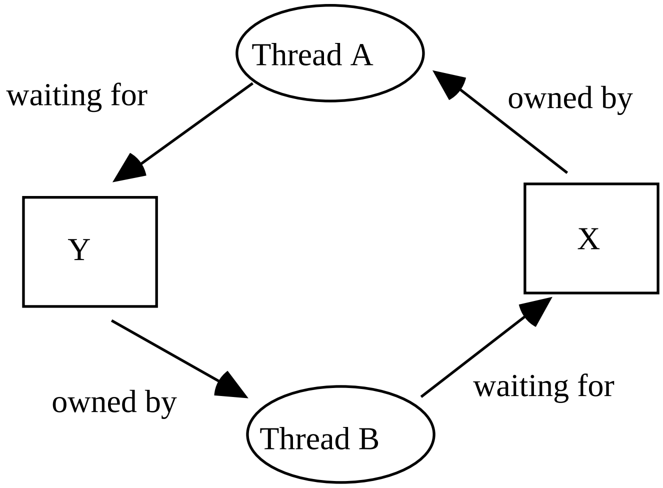
1. Limited access (for example: mutex or bounded buffer)

2. No preemption (if someone has resource, can't take it away)

3. Multiple independent requests -- "wait while holding"

4. Circular chain of requests

Can draw graph to see if you have a circular chain:



Example of deadlock

10.3 Solutions to Deadlock

10.3.1 Detect deadlock and fix

```
scan graph
detect cycles
fix them       // this is the hard part!
```

a) Shoot thread, force it to give up resources.
This isn't always possible -- for instance, with a mutex, can't shoot a thread and leave world in a consistent state.

b) Roll back actions of deadlocked threads (transactions)

Common technique in databases.

10.3.2 Preventing deadlock

Need to get rid of one of the four conditions.
a) Infinite resources

b) No sharing -- totally independent threads.

c) Don't allow waiting -- how phone company avoids deadlock

d) Preempt resources

Example: Can preempt main memory by copying to disk

e) Make all threads request everything they'll need at beginning

If you need 2 chopsticks, grab both at same time.

Problem is -- predicting future is hard, tend to over-estimate resource needs (inefficient)

Banker's algorithm: more efficient than reserving all resources on startup

1. State maximum resource needs in advance
2. Allocate resources dynamically when resource is needed -- wait if granting request would leads to deadlock (request can be granted if some sequential ordering of threads is deadlock free)

Bankers algorithm allows the sum of maximum resource needs of all current threads to be greater than the total resources, as long as there is some way for all the threads to finish without getting into deadlock.

For example, you can allow a thread to proceed if
total available resources - # allocated >= max remaining that might be needed by this thread in order to finish and release its resources (guarantees this thread can finish)

Example of Banker's algorithm with dining lawyers: chopsticks in middle of table.

Deadlock free if when try to grab fork, take it unless it's the last one, and no one would have 2.

What if k-handed lawyers?

Deadlock free if when try to grab fork: take it unless

it’s the last one, and no one would have k

it’s the next to the last, and no one would have k-1,

...

f) Make everyone use the same ordering in accessing resources.

For example, all threads must grab semaphores in the same order

(x.P; y.P)

Typically, a combination of techniques!