

# CS 162 Operating Systems and Systems Programming

Professor: Anthony D. Joseph  
Spring 2001

## Lecture 10: Deadlock

### 10.0 Main Points:

- Definition of deadlock
- Conditions for its occurrence
- Solutions for breaking and avoiding deadlock

Solutions pose a dilemma:

- Simple solutions – inefficient
- Complex solutions – inefficient and unpleasant

### 10.1 Definitions

#### 10.1.1 Resources

**Threads** – active

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

Two kinds of resources:

- **Preemptable** – can take it away (CPU)
- **Non-preemptable** – must leave with thread  
(disk space – what would you think if I took space for your files?)

**Mutual exclusion** – the right to enter a critical section is a kind of resource

#### 10.1.2 Starvation vs. Deadlock

**Starvation** – thread waits indefinitely (for example, because some other threads are using resource)

**Deadlock** – circular waiting for resources

Example:

- I have resource A and need resource B to get my job done.

- You have resource B and need resource A to get your job done.

=> *Deadlock implies starvation, but not vice versa*

- Starvation can end (but doesn't have to)
- Deadlock can't end without external intervention.

**Deadlock need not be deterministic:**

X = 1; y = 1;

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

Deadlock won't always happen with this code, but it might.

## 10.2 Conditions for deadlock

### 10.2.1 Motivation

Deadlock can happen with any kind of resource.

Deadlocks can occur with multiple resources. Means you can't decompose the problem – can't solve deadlock for each resource independently.

For example:

1. one thread grabs the memory it needs
2. another grabs disk space
3. another grabs the tape drive

Each waits for the other to release.

**Deadlock can occur whenever there is waiting.**

Example: dining lawyers

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

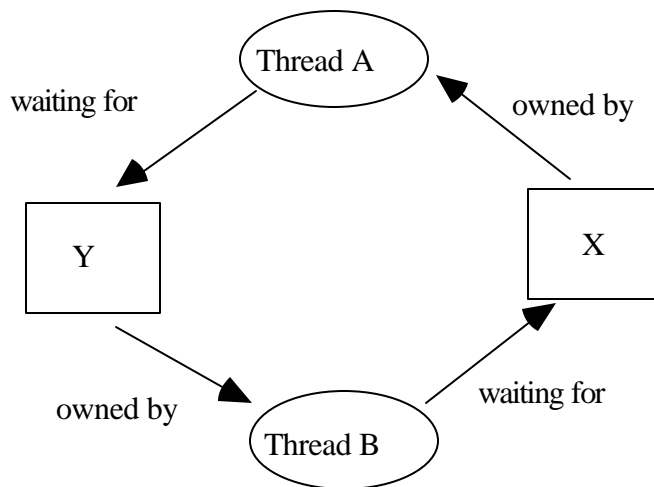
What if all grab at the same time? Deadlock.

## 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

IF one can preempt resources, then one option is to let deadlocks happen and fix things up after the fact.

```
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

What if you can't reclaim a deadlocked resource?

Then you need to prevent deadlocks from ever happening in the first place.

**Need to get rid of one of the four conditions that allow deadlock to occur.**

- 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 lead 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:

The total available resources – # allocated  $\geq$  max remaining that might be needed by any thread.

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, systems employ a combination of techniques!

## 10.4 Summary

When multiple threads or processes contend for shared resources then starvation and deadlock can occur.

If its OK to preempt resources held by a deadlocked process then one can let deadlocks happen and fix things up after the fact. If deadlocks happen infrequently then this provides an efficient common case solution in exchange for a (hopefully) acceptable exception case cost.

If one can't preempt resources held by a deadlocked process then deadlocks have to be prevented from happening in the first place. That is, one has to prevent one or more of the four conditions needed to allow deadlock to occur from holding.