Deadlock

Mendel Rosenblum

Why do we have multiple locks?

- Reduce contention (fine grain locking vs coarse grain locking)
- Enable modularity (lock per structure)
- Threads often need multiple locks simultaneously

Hazard: **Deadlock**

Simple Deadlock Example

```
std::mutex m1;
std::mutex m2;
```

Thread A:

```
m1.lock();
m2.lock();
...
m2.unlock();
m1.unlock();
```

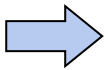
Thread B:

```
m2.lock();
m1.lock();
...
m1.unlock();
m2.unlock();
```

Simple Deadlock Example

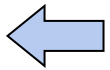
```
std::mutex m1;
std::mutex m2;
```

Thread A:



```
m1.lock();
m2.lock();
...
m2.unlock();
m1.unlock();
```

Thread B:

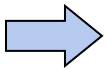


```
m2.lock();
m1.lock();
...
m1.unlock();
m2.unlock();
```

Simple Deadlock Example

```
std::mutex m1;
std::mutex m2;
```

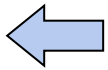
Thread A:



```
m1.lock();
m2.lock();
...
m2.unlock();
m1.unlock();
```

Thread B:

```
m2.lock();
m1.lock();
...
m1.unlock();
m2.unlock();
```



Informal definition of deadlock

- A collection of threads are all blocked
- Each thread is waiting for a resource owned by one of the other threads
- Since all threads are blocked, none can release their resources

English dictionary definition of word **deadlock**:

mutual blockage preventing progress

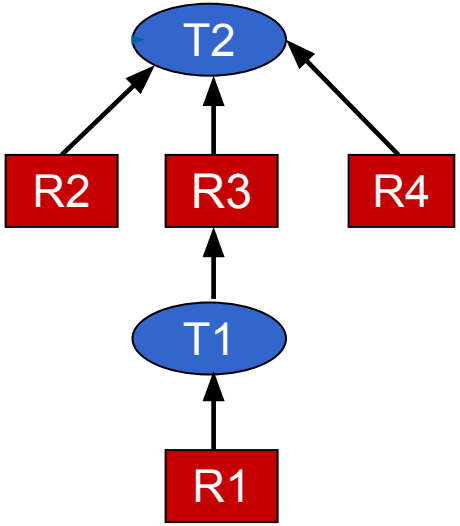
Aside: Theory and Systems

- Lots of interesting computer science theory
  - Not much has been super helpful to operating systems
  - Deadlock is an exception

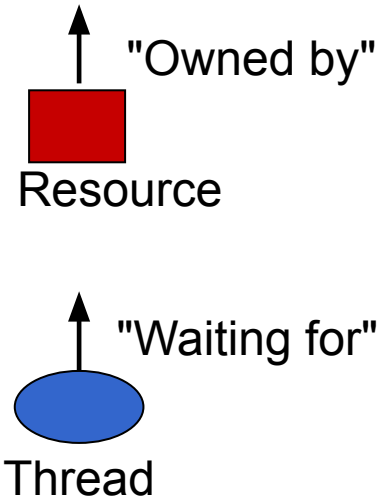
Four Conditions for Deadlock

- Limited access: resources cannot be shared.
  - Mutual exclusion
- No preemption
  - Once given, a resource cannot be taken away
- Multiple independent requests
  - Threads don't ask for resources all at once
  - Hold resources while waiting
- A circularity in the graph of requests and ownership
  - Circular Wait

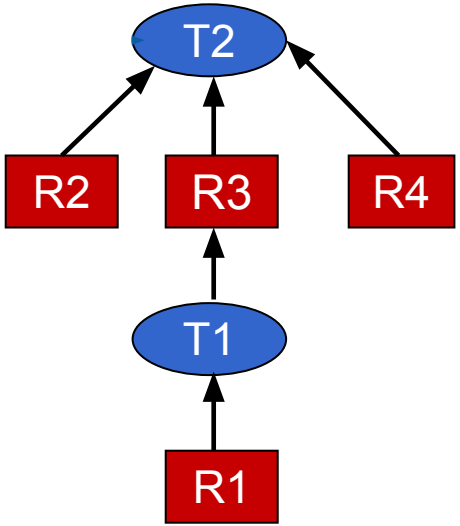
Circular Requests



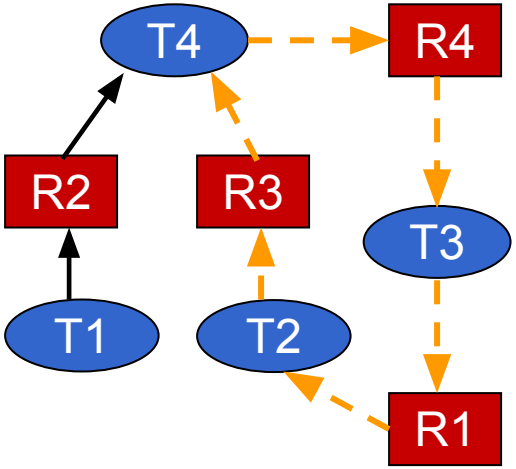
No Circularity



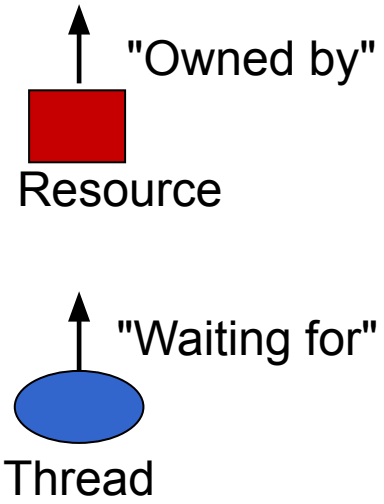
Circular Requests



No Circularity



Circularity



Deadlock Is Broader Than Locks

- Deadlock can occur over anything that causes waiting:
  - Locks (Discrete resource)
  - Memory exhaustion (Continuous resource)
  - Tape drives
  - Network messages
  - Distributed systems
- In general, don't know in advance which resources a thread will need

Solution #1 – Deadlock Detection

- Determine when system is deadlocked
- Break the deadlock by terminating one of the threads
- Usually not practical in operating systems
  - But often used in database systems where transactions can be aborted and retried

Solution #2 – Deadlock Prevention

Eliminate one necessary condition

- **Limited access**: Threads never have to wait
  - Create enough resources so thread never wait
  - Not reasonable for most applications (e.g Locks)
- **No preemption:** Just take resources away
  - Works for some resources (e.g. CPUs) not others (e.g. Locks)
- **Multiple independent requests:** Request all resources up front
  - Require threads to request all resources at the same time; either get them all or wait for them all
    - Tricky to implement: must wait for several things without locking any of them
    - Inconvenient for thread: hard to predict needs in advance; May lead to over-allocation.
- **Circular Wait:** Order resource allocation to avoid cycles
  - Prevent circularities: all threads request resources in the same order (e.g., always lock l1 before l2).
  - This is the most common approach used in operating systems

Prevent Circular Wait (Practical Approach)

- Impose global lock ordering
  - All threads acquire locks in same order
- Assignment a number to each lock (i.e. lock rank)
  - Acquire locks always in increasing (or always decreasing) order
  - Can check lock rank while running

Case Study: Two Processes doing `mv` commands

Process #1

Command: **`mv a/x b/y`**

Process #2

Command: **`mv b/z a/q`**

Consider locking in parameter order:

Lock dir a
Lock dir b

Lock dir b
Lock dir a

How can we avoid deadlock?

Design Insight

- Deadlock is a global design problem
  - Breaks modularity
- Need agreement among all modules about the order of locks
  - Easy to reintroduce deadlocks when changing the system
- Can be painful: Need to get a resource but can't due to ordering