EECS 482
Introduction to Operating Systems
Winter 2019
Manos Kapritsos

Thanks to Harsha Madhyastha and Peter Chen for the slides and notes
Deadlock

- Over-constrained synchronization:
  - Cycle of threads holding some resources and waiting on some resources held by others

```
thread A
resource 1
  resource 2
  thread B
```
Preventing deadlock

- What if we don’t grant resources that will lead to cycle in waits-for-graph?

The deadlock may be unavoidable before the cycle forms.
Four necessary conditions for deadlock

- Limited resources
  - Not enough to serve all threads simultaneously

- No preemption
  - Can’t force threads to give up resources

- Hold and wait
  - Threads hold resources while waiting to acquire other resources

- Cyclical chain of requests
Eliminating circular chain

- Impose global ordering of resources
Eliminating hold-and-wait

- Two ways to avoid hold and wait:
  - Wait for all resources needed to be free; grab them all atomically
  - If cannot get a resource, release all and start over

- Move resource acquisition to beginning
  
  Phase 1: acquire all resources
  
  Phase 2: while (!done) {
      work
  }
  
  Phase 3: release all resources

- Ensures working threads will complete
Banker’s algorithm

- An alternative solution to eliminate hold-and-wait
  - Allows for more concurrency
- Declare resources at the beginning, but don’t actually acquire them
  
  Phase 1: `declare` all resources
  Phase 2: `while (!done) {
            acquire resource `if safe`
            work
          }
  Phase 3: `release all resources`
Banker’s algorithm

Phase 1: declare all resources
Phase 2: while (!done) {
    acquire resource if safe
    work
}
Phase 3: release all resources

- Only grant resource if it’s “safe”, otherwise block
  - Safe means I can guarantee that all threads can finish

- Criterion: Can I grant the maximum resources of all threads in some sequential order?
The bank example

● A bank has $6000

● Customers establish credit limit, and then can borrow money (up to their credit limit)
  ♦ Credit limit is “max resource usage”

● When their business is done, customers return the money
Bank solution #1

- Bank gives money upon request, if it’s available

- Example:
  - Ann asks for credit of $2000
  - Bob asks for credit of $4000
  - Charlie asks for credit of $6000

- Can the bank approve all these credit lines?

- No, a deadlock can form:
  - Ann borrows $1000
  - Bob borrows $2000
  - Charlie borrows $3000
Bank solution #1

- Bank gives money upon request, if it’s available
- Only works if the bank reserves the money when credit line is established. Customers may have to wait at the credit approval stage.
- Example:
  - Ann asks for credit of $2000. Approved.
  - Bob asks for credit of $4000. Approved.
  - Charlie asks for credit of $6000. Must wait until Ann and Bob drop their lines of credit.
Banker’s algorithm

- **Bank approves all credit requests. Bank gives money upon request but only if it’s safe.**

- **Example:**
  - Ann borrows $1000 (bank has $5000 left)
  - Bob borrows $2000 (bank has $3000 left)
  - Charlie wants to take out $2000. Is this allowed?

- **It is allowed iff there is some sequential ordering of fulfilling everyone’s max resources**
  - Charlie leaves bank with $1000.
  - Ann can finish, leaving $2000
  - So Bob can finish, leaving $4000
  - So Charlie can finish.
Banker’s algorithm

- **Bank approves all credit requests. Bank gives money upon request but only if it’s safe.**
- **Example #2:**
  - Ann borrows $1000 (bank has $5000 left)
  - Bob borrows $2000 (bank has $3000 left)
  - Charlie wants to take out $2500. Is this allowed?
- It is allowed iff there is some sequential ordering of fulfilling everyone’s max resources
- $500 left in bank. Can’t guarantee that any single customer would finish.
Banker’s algorithm

- **Bank approves all credit requests. Bank gives money upon request but only if it’s safe.**

- **Example #3:**
  - Ann borrows $1000 (bank has $5000 left)
  - Bob borrows $2000 (bank has $3000 left)
  - Charlie wants to take out $2000. Is this allowed?

- **$1000 left in bank.**
  - Ann can finish, leaving $2000 in bank
  - Bob might need $3000 more
  - Charlie might need $4000 more
  - Can’t guarantee that either can finish
Banker’s algorithm for the dining philosophers

- Max resource need for each philosopher is 2
- Grant chopstick unless it’s the last chopstick and nobody has 2
Banker’s algorithm

- Allows system to overcommit resources
  - Sum of max resources can be greater than total resources

- Elegant algorithm, when applicable
  - Sometimes it’s hard to know what the max resource need is
  - Mostly applies to “quantitative” (fungible) resources
    » Typically not applicable to locks
Project 2

is due in 1 week!

- Advice:
  - Test your code *while* writing it
  - Write your test cases *before* the code
  - Go through the spec, write a test case for *every* condition required by the spec
Threads and Concurrency

- Concurrent programming using threads simpler than event-based programming
- Threads must synchronize access to shared data
- Over-constrained synchronization $\rightarrow$ deadlock
Midterm

- March 1st, 10am-12pm
- Covering everything up to (and including 😊) this slide
- Study the material, study your p2 code
- Sample midterm posted, one more to come
- Contact me if you have a conflict or extra time request and have not received an email from us