ECE 570/670

David Irwin

Lecture 4

Administrative Details

- Web: https://courses.umass.edu/eceng570-deirwin/spring23
  - Username: **irwin-ece**, Password: **myclass**

- Assignment 0 is now posted (see calendar for due date)
  - Look at Assignment 0, find assignment partner (if desired), request username/password, start assignment
  - Will register new groups with autograder over weekend
- UNIX paper assigned for next class

ECE670 Paper Reviews

- Paper reviews starting now
  - Should do 10 of 12 for class

- Sign up for account linked of Assignments page
  - Only 7 currently signed up
  - Everyone must sign up before I can assign reviews!

- Paper Review
  - Briefly summarize Paper
  - What are the paper's strengths and contributions?
  - Why do you think this paper is considered novel/important?
    - Most of these papers are seminal, so they have few weaknesses
    - They were already accepted at the top venues

- See: https://people.inf.ethz.ch/troscoe/pubs/review-writing.pdf

Assignment 0

- **Inverted Index**
  - Virtual appliance posted (see Resources page)
  - C/C++ references posted (also see Resources page)
    - …and Google
  - Autograder information posted
    - See link at bottom of Assignment 0 page

- **Configuration**
  - Need to know assignment partners ASAP
    - Will use them to configure the autograder
  - 7 people have yet to sign up

Assignment 0

- **Inverted Index**
  - ***Input:*** one argument → a file name
    - Contains list of other text file names
    - Each text file has words, i.e., ASCII characters, in it
  - ***Output:*** all words in alphabetical order, followed by document numbers in which they appear
- **Example**
  - ./inverter index.txt
  - index.txt contains foo1.txt and foo2.txt
    - foo1.txt contains "this is a test. cool."
    - foo2.txt contains "this is also a test.\nboring."

Assignment 0

- **Inverted Output**

  a: 0 1
  also: 1
  boring: 1
  cool: 0
  is: 0 1
  test: 0 1
  this: 0 1

  foo1.txt → Document 0
  foo2.txt → Document 1

- **To compile:** $> g++ inverter.cc –o inverter
- **To run:** $> ./inverter
- Debug using gdb – see web for tutorials

- Free to develop anywhere, but **must run and submit** from virtual appliance

Assignment 0

- **Virtual Appliance**
  - Download ECE670-Spring23.ova file from Resources page
    - Requires 2GB of space; will send out passwords via email
  - Install VirtualBox (works with Windows, Mac, Linux)
  - Click "File"; then "Import Appliance"; click *.ova file
  - Start VM and login with supplied username/password
- **Autograder**
  - ***The autograder is merciless! Start early!***
  - 1 submission per day plus 3 bonus submissions
  - Receive email of results to @umass addr within 4 hours
  - To submit: ***$> submit570 0 inverter.cc***
    - Make sure you are connected to the Internet first

Assignment 0

- **Reminder: do not put code on public online repository, e.g., GitHub, BitBucket, Google Code, etc.!**

- **Assignment 0 is meant to be easy**
  - Do not be lulled into a false sense of security
  - Assignments get progressively more challenging
  - Purpose of this assignment:
    - 1. Understand how the autograding system works
    - 2. Get a refresher on coding in C/C++

Assignment 0

- **Pro Tip #1**
  - Use "scp" to copy files to/from your VM
    - scp == **S**ecure **C**o**p**y
  - **In Mac:**
    - 1. Open Applications->Utilities->Terminal
    - 2. scp –P 2500 file.txt group<#>@localhost:.
    - 3. scp –P 2500 group<#>@localhost:file.txt .
    - Will need to enter username and password
  - **In Windows:**
    - Use WinSCP program posted on website
    - Use Windows Shell

Assignment 0

- **Pro Tip #1 (cont'd)**
  - "Port forwarding" via port 2500 should already be setup
    - This is why you put the "-P 2500" arguments after scp
  - If scp doesn't work, and port forwarding is not setup, there are directions posted on the Resources part of the web site to configure it

Assignment 0

- **Pro Tip #2**
  - If your VM is too slow, don't use the GUI
  - **ssh** into it from your laptop instead
    - ssh = **S**ecure **Sh**ell
  - **In Mac:**
    - Open Applications->Utilities->Terminal
    - Run $>ssh –p 2500 group<#>@localhost
    - Enter username and password
  - **In Windows:**
    - Use Putty program posted on web or…
    - Windows Shell

670 Projects

- **Proposals due Friday March 24th**
  - Requirements on website (2 page proposal)
    - Must have a **software implementation** component
    - Must have a **performance evaluation** component
  - Form into groups (up to 4) and send me names
    - Will post a calendar for group meetings

- **Stock project:** design a web server
  - Implement threaded and/or event-driven web server
  - Evaluate and compare their performance

Last Time

- Discussed events and threads

Why Events Are A Bad Idea (for high-concurrency servers)

- Conventional Wisdom
  - Threads "easy" to program….
  - …but don't perform well at scale
    - E.g., *with 100,000 threads*
- Why?
  - Performance, Control flow, Synchronization, State Management, Scheduling

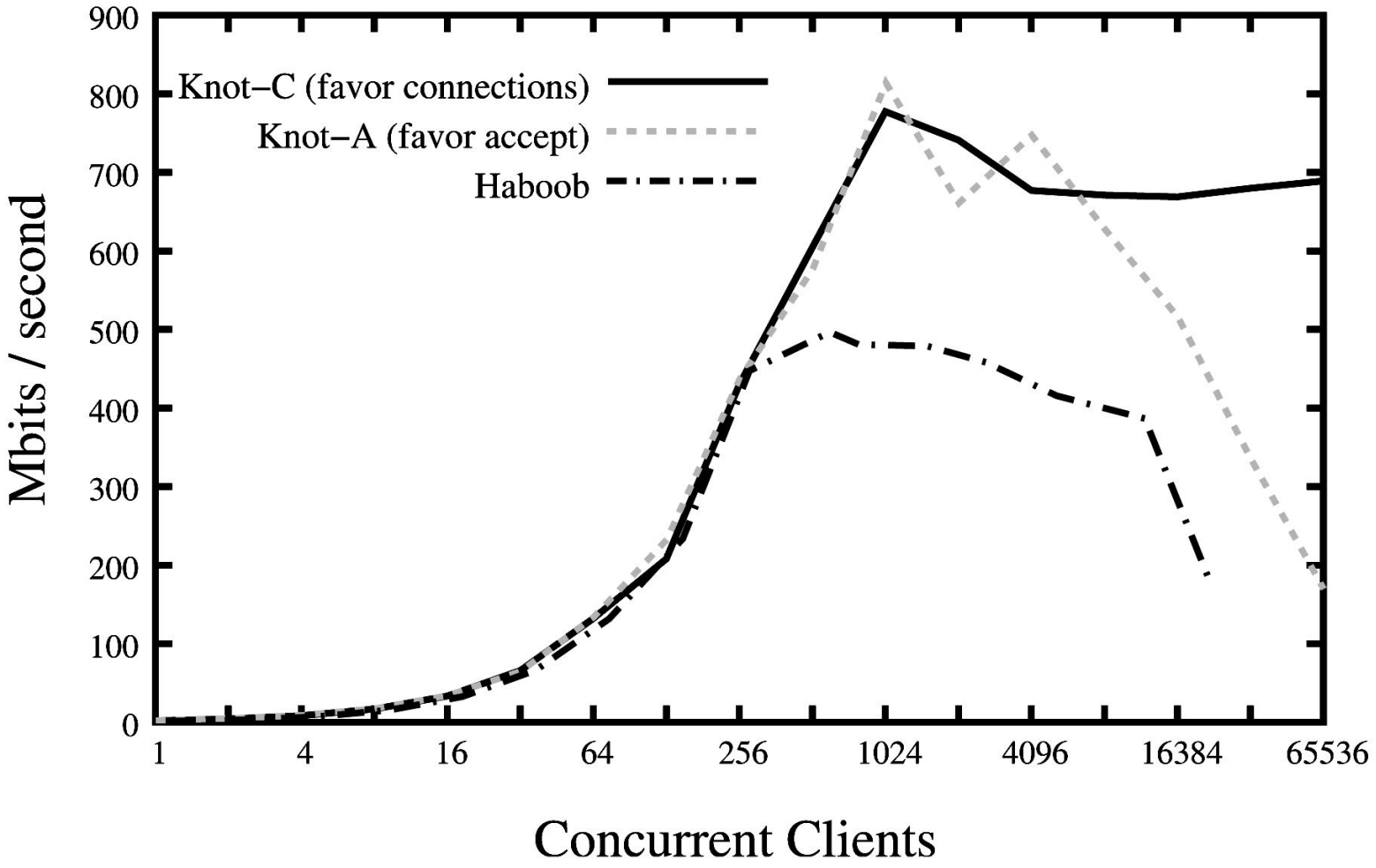
Why Events Are A Bad Idea (for high-concurrency servers)

- Performance – remove O(n) operations
- Control Flow – no need for complex non-linear flows
- Synchronization – not easy for either on multiprocessors
- State Management – stacks easier to manage
- Scheduling – scheduling tricks applicable to threads

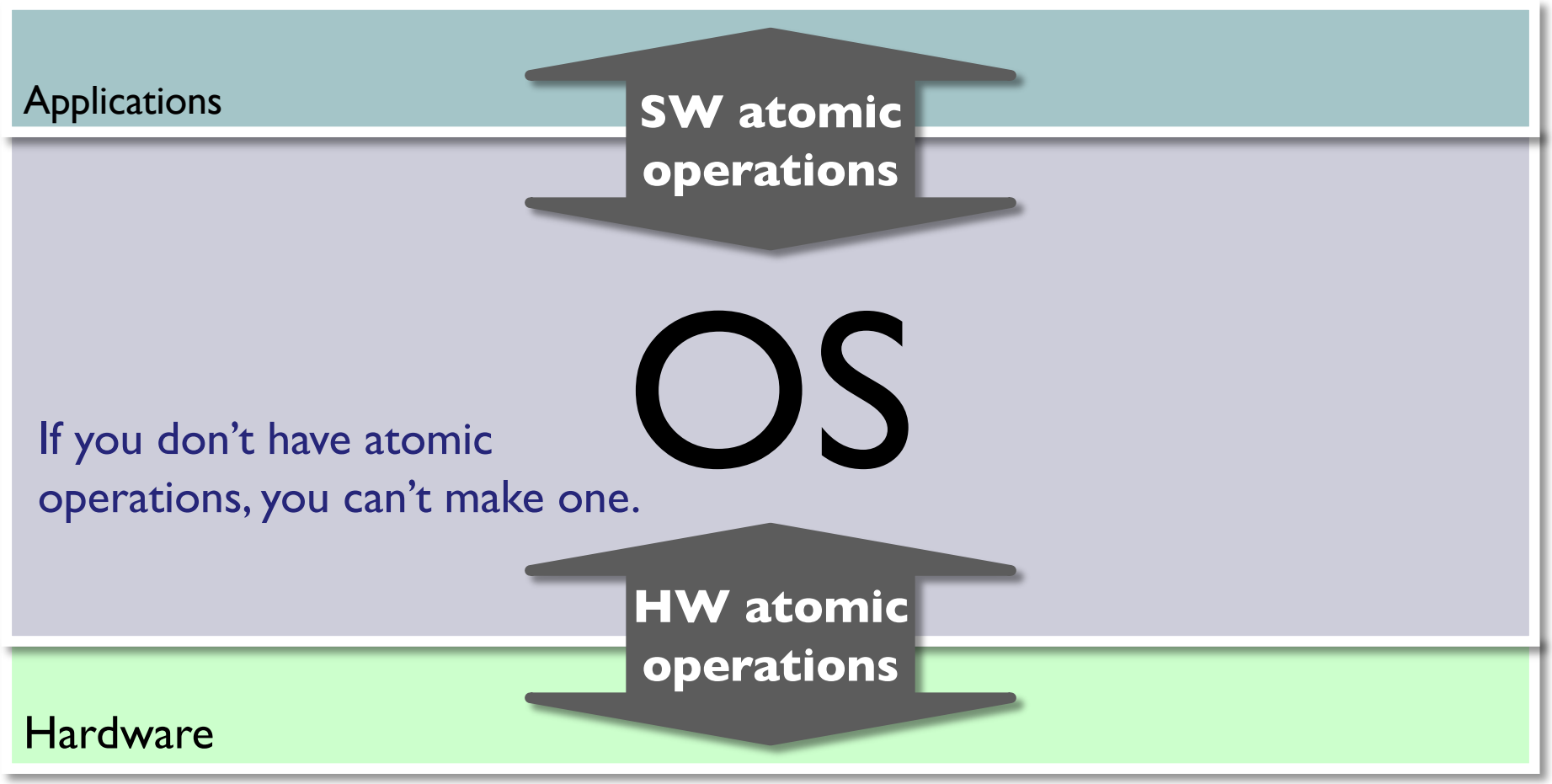
Why Events Are A Bad Idea (for high-concurrency servers)

- Directions:
  - Dynamic Stack Growth (change stack size dynamically)
  - Live State Management (automatically purge stack of unnecessary state)
  - Synchronization tools (use automated tools to find synchronization bugs)

Why Events Are A Bad Idea (for high-concurrency servers)



Virtual/physical interfaces



Debugging non-determinism

- Requires **worst-case** reasoning
  - Eliminate **all** ways for program to break
- Debugging is hard
  - Can't test all possible interleavings
  - Bugs may only happen sometimes

- **Heisenbug**
  - Re-running program may make the bug disappear
  - Doesn't mean it isn't still there!

Constraining concurrency

- **Synchronization**
  - Controlling thread interleavings
- Some events are independent
  - Don't depend on shared state
  - Relative order of these events don't matter
- Other events are dependent
  - Output of one can be input to another
  - Their order can affect program results

Goals of synchronization

1. All interleavings must give correct result
   - Correct concurrent program
     - Works no matter how fast threads run
   - Important for your projects!
2. Constrains program as little as possible
   - Why?
     - Constraints slow program down
     - Constraints create complexity

“Too much milk” principal

"Too much milk" rules

- The fridge must **always** be stocked with milk
  - Milk expires quickly, so never >1 milk. We need exactly 1.
- David and Jeannie (two threads)
  - Can come home at any time
  - If either sees an empty fridge, must immediately go to store and buy milk
  - Code below (no synchronization)

```
if (noMilk){
  buy milk;
}
```

Unsynchronized code will break

Time		
3:00	Look in fridge (no milk)	
3:05	Go to grocery store	
3:10		Look in fridge (no milk)
3:15	Buy milk	
3:20		Go to grocery store
3:25	Arrive home, stock fridge	
3:30		Buy milk
3:35		Arrive home, stock fridge Too much milk!

What broke?

- Code worked sometimes, but not always
  - Code contained a **race condition**
  - Processor speed caused incorrect result

- First type of synchronization
  - **Mutual exclusion**
  - **Critical sections**

Synchronization concepts

- **Mutual exclusion**
  - Ensure only one thread doing something at a time
    - *i.e., one person shops at a time*
  - Code blocks atomic with respect to each other
  - Threads can't run code blocks at same time

Synchronization concepts

- **Critical section**
  - Code "block" that must run atomically
    - "with respect to some other pieces of code"
- If A and B are critical with respect to each other
  - Threads must not interleave events in A and B
  - A and B *mutually exclude* each other
- Often conflicting code is in the same block
  - But executed by different threads
  - Reads/writes of shared data (e.g. screen, fridge)

Back to "Too much milk"

- What is the critical section?

```
if (noMilk){
  buy milk;
}
```

- Jeannie and David's critical sections
  - Must be atomic with respect to each other

"Too much milk" Solution 1

- Assume only atomic load/store
  - Build larger atomic section from load/store
- **Idea:**
  1. Leave notes to say you're taking care of it
  2. Don't check milk if there is a note

Solution 1 code

- Atomic operations
  - *Atomic load:* check note
  - *Atomic store:* leave note

```
if (noMilk) {
  if (noNote){
    leave note;
    buy milk;
    remove note;
  }
}
```

Does it work?

```
if (noMilk) {
  if (noNote){
    leave note;
    buy milk;
    remove note;
  }
}
```

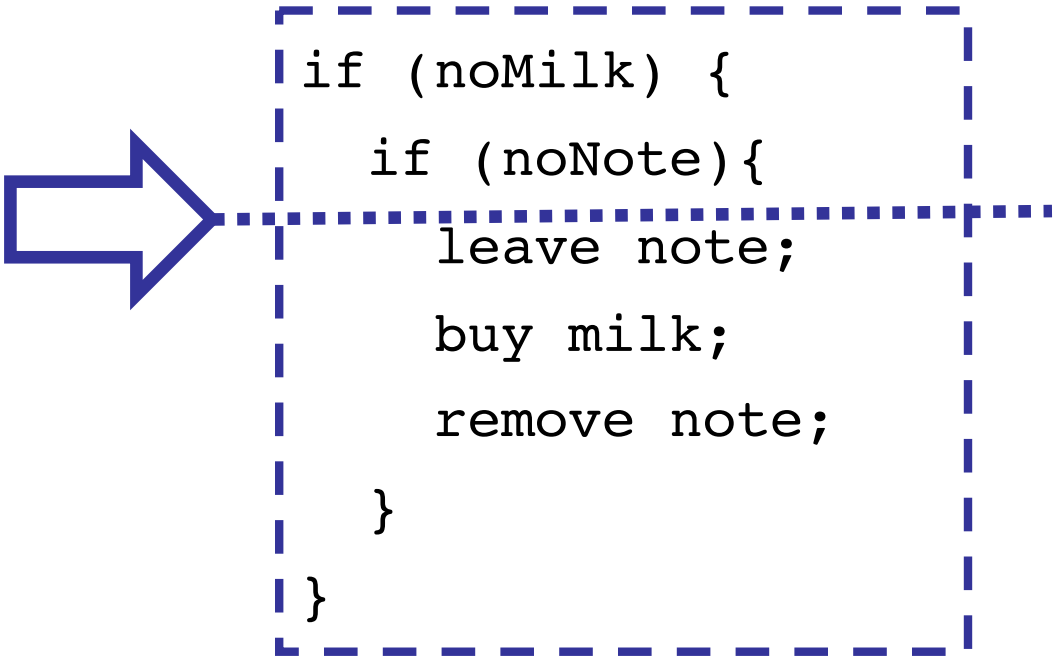
```
if (noMilk) {
  if (noNote){
    leave note;
    buy milk;
    remove note;
  }
}
```

Is this better than no synchronization at all?

What if "if" sections are switched?

What broke?

- David's events can happen
  - After Jeannie checks for a note, but…
  - before Jeannie leaves a note



Next solution

- **Idea:**
  - Change the order of “leave note”, “check note”
  - Requires labeled notes
  - Why?
    - (You’ll see your note and never do anything)

Does it work?

```
leave noteJeannie
if (no noteDavid){
   if (noMilk){
     buy milk;
   }
}
remove noteJeannie
```

```
leave noteDavid
if (no noteJeannie){
   if (noMilk){
     buy milk;
   }
}
remove noteDavid
```

Nope. Why?
Illustration of "starvation."

What about now?

```
while (noMilk){
   leave noteJeannie
   if(no noteDavid){
     if(noMilk){
       buy milk;
     }
   }
   remove noteJeannie
}
```

```
while (noMilk){
   leave noteDavid
   if(no noteJeannie){
     if(noMilk){
       buy milk;
     }
   }
   remove noteDavid
}
```

Nope. (Same starvation problem as before)

Next solution

- We're getting closer
- Problem
  - Who buys milk if both leave notes?
- Solution
  - Let Jeannie hang around to make sure job is done

Does it work?

```
leave noteJeannie
while (noteDavid){
  do nothing
}
if (noMilk){
  buy milk;
}
remove noteJeannie
```

```
leave noteDavid
if (no noteJeannie){
  if (noMilk){
    buy milk;
  }
}
remove noteDavid
```

Yes! It does work! Can you prove it?

Downside of solution

- Complexity
  - Hard to convince yourself it works
- Asymmetric
  - Jeannie and David run different code
- Not clear if this scales to > 2 people
- Jeannie consumes CPU while waiting
  - `while (noteDavid) { do nothing }`
  - **Busy-waiting**
- **But:** only needed atomic load/store!

Raising the level of abstraction

- Mutual exclusion with atomic load/store
  - Painful to program
  - Wastes resources (busy waiting)
    - Need more HW support to fix this
    - Will be covered later
- OS can provide higher level abstractions