ECS 251: Cooperating threads

Sam King

Administrative

- HW 1 out last Tuesday, due on 1/22
- Quiz 1 today

- I’m out of town Thurs next week, TBD on how we’ll handle it

Project groups due on Tues

- Please make sure that you've formed a group and that you submit your group name and group members next week

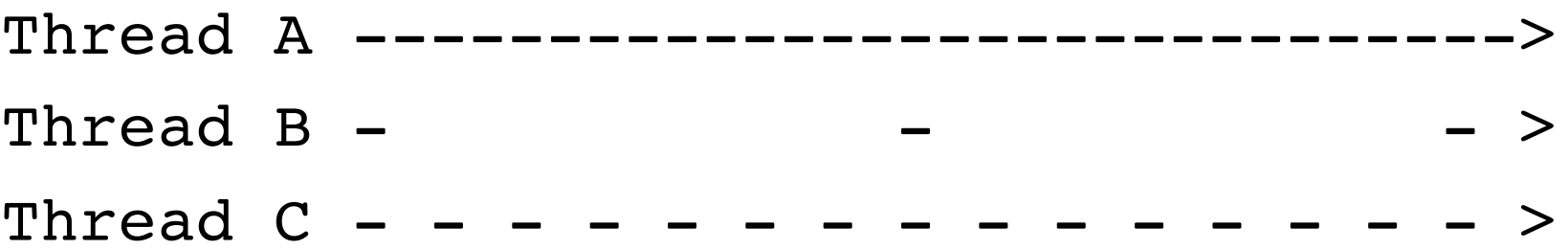
- A few project ideas have come through via email, happy to provide feedback

Cooperating threads

- How multiple threads can cooperate on a single task
  - Assume for now that we have enough physical processors for each thread
  - Later we'll discuss how to give this illusion of infinite physical processors on a single processor

Ordering of events

- Ordering of events from different threads is non-deterministic
  - Processor speeds may vary
  - E.g., after 10 seconds, different thread have different amounts of work done



Non-determinisim

- Non deterministic ordering produces non deterministic results

- Printing example
  - Thread A: print ABC
  - Thread B: print 123
  - Possible outputs?

  - Impossible outputs? Why or why not?

  - What is being shared?

Arithmetic example

- Initially y=10
- Thread A: x = y+1;
- Thread B: y = y*2;

- Possible results?

Atomic operations

- Example
  - Thread A: x=1;
  - Thread B: x=2;
  - Possible results?

  - Is 3 a possible output?

Atomic operations

- Before we can reason **at all** about cooperating threads, we must know that some operation is **atomic**

- Atomic: indivisible. Either happens in its entirety without interruption, or has yet to happen at all.
  - No events from other threads can happen in between the start and the end of an atomic event

Example disc.

- In assignment example above, if assignment to x is atomic, then only possible results are 1 and 2.

- In print example above, what are the possible output if each print statement is atomic?

- In print example, assuming printing a char was atomic. What if printing a single char was **not** atomic?

Atomicity disc.

- On most machines, memory load and store are atomic
- But, many instructions are **not** atomic
  - Floating point store on 32-bit machine
- If you don't have any atomic operations, you can't make one
  - Fortunately, H/W designers have helped us out…

Another example

```
Thread A                    Thread B
i=0                         i=0
while(i<10) {               while(i>-10) {
    i++                         i--
}                           }
Print “A finished”          Print “B finished”
```

- Who will win?
- Is it guaranteed that someone will win?
- What if threads run at exactly the same speed and start close together?
- What if i++ and i-- are not atomic?

I++ I-- not atomic

```
Tmp (private) = I + 1;
I = Tmp;

(A) TmpA = I + 1 (I.e. 1)
(B) tmpB = I - 1 (I.e. -1)
(A) I = tmpA
(B) I = tmpB
```

Another example disc. cont.

- Should you worry about this happening?

- Non-deterministic interleaving makes debugging challenging
  - Heisenbug

Synchronizing between multiple threads

- Must control interleaving between threads
  - Order of some operations irrelevant
    - Independent
  - Other operations are dependent and order does matter
- All possible interleaving must yield a correct answer
  - A correct concurrent program will work no matter how fast the processors are that execute the various threads

Synchronizing between multiple threads

- All interleavings result in correct answer
- Try to constrain the thread executions as little as possible
- Controlling the execution and order of threads is called “synchronization”

Too much milk

- Problem definition
  - Sam and Anne want to keep refrigerator stocked with at most one milk jug
  - If either sees fridge empty, she/he goes to buy milk
  - Correctness properties:
    - Someone will buy milk if needed
    - Never more than one person buys milk

Solution #0 (no sync)

```
Sam:
if(noMilk) {
    buy milk
}
```

```
Anne:
if(noMilk) {
    buy milk
}
```

```
      Sam                          Anne
3:00 look in fridge
       (no milk)
3:05 go to Safeway
3:10                          look in fridge (no milk)
3:15 buy milk
3:20                          go to Safeway
3:25 arrive home, add milk
3:30                          buy milk
3:35                          arrive home, add milk
                                Too Much Milk!
```

Mutual exclusion

- Ensure that only 1 thread is doing a certain thing at one time
  - Only one person goes shopping at one time
- Critical section
  - A section of code that needs to run atomically w.r.t. other code
  - If code A and code B are critical sections w.r.t. each other
    - Threads cannot interleave events from A and B
  - Critical sections must be atomic w.r.t. each other
    - Share data (or other resourced, e.g., screen, fridge)
- What is the critical section in solution #0?

Too much milk (solution #1)

- Assume only atomic operations are load and store
- Idea: leave note that going to check on milk status

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

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

- Does this work? If not, when could it fail?

- Is solution #1 better than solution #0?

Too much milk (solution #2)

- Idea: change the order of “leave note” and “check note”.
  - Labeled notes

```
Sam:
leave noteSam
if (no noteAnne) {
    if( noMilk ) {
        buy milk
    }
}
remove noteSam
```

```
Anne:
Leave noteAnne
if (no noteSam) {
    if( noMilk ) {
        buy milk
    }
}
remove noteAnne
```

Solution #2 disc.

- Does solution #2 work? If not, when could it fail?

Too much milk (solution #3)

•Idea: have a way to decide who will buy milk when both leave notes at the same time. Have Sam hang around to make sure job is done.

```
Sam:
leave noteSam
while (noteAnne) {
    do nothing
}
if (noMilk) {
    buy milk
}
remove noteSam
```

```
Anne:
leave noteAnne


if (no noteSam) {
    if( noMilk ) {
        buy milk
    }
}
remove noteAnne
```

Too much milk (solution #3)

- Sam's "while(noteAnne)" prevents him from running his critical section at the same time as Anne's
- Proof of correctness
  - "Exercise to the reader"
- Correct, but ugly
  - Complicated
  - Asymmetric
  - Inefficient
    - Sam consumes CPU time while waiting (**Busy Waiting**)