Operating Systems Design (CS 423)

Elsa L Gunter
2112 SC, UIUC
http://www.cs.illinois.edu/class/cs423/

Based on slides by Roy Campbell, Sam King, and Andrew S Tanenbaum

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 illusion of infinite processors on 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

Thread A -->
Thread B -   -   -   - >
Thread C - - - - - - - - - - - - - - - >

Nondeterminisim

- 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?

Non-interference and Atomic operations

- Non-interference assumption:
  - Before we can reason **at all** about cooperating threads, we must know that some operations will execute to completion without interference from any other source
  - Non-interference: operation will always return result as if it were the only operation running
- Operation that in all circumstances will execute to completion without interference is **atomic**

Previous Examples

- 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 were **not** atomic?

Atomicity

- 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, difficult to make one (bakery algorithm
  - 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

- Should you worry about this happening?
- Non-deterministic interleaving makes debugging challenging

Synchronizing 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 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

- The Gunter household drinks a lot of milk, but has a small fridge
- Problem: Carl and Elsa want there to always at least one gallon of milk in the fridge for dinner; fridge holds at most two gallons
- If either sees there is less than one gallon, goes to buy milk
- Specification: Someone buys milk if running low
- Never more than two gallons of milk milk

Solution #0 – no sync

Carl	Elsa
5:30 Comes home	
5:35 Checks milk	
5:40 Goes to store	
5:45	Comes home
5:50 Buys milk	Checks milk
5:55 Goes home	Goes to store
6:00 Puts milk in Fridge	Buys milk
6:05	Comes home
6:10	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
- Carl: if (no note)
  {if (milk low) {leave note; buy milk; remove note;}
- Elsa: if (no note)
  {if (milk low) {leave note; buy milk; remove note;}
- What can go wrong?
- Is this better than before?

Too much milk (solution #2)

- Idea: Change order of leave note and check milk
- Carl: if (milk low)
  {if (no note) {leave note; buy milk; remove note;}
- Elsa: if (milk low)
  {if (no note) {leave note; buy milk; remove note;}
- What can go wrong?
- Is this better than before?

Too much milk (solution #3)

- Idea: Protect more actions with note
- Carl: if (no note) {leave note;
  if (milk low) {buy milk}; remove note;}
- Elsa: if (no note) {leave note; if (milk low)
  {buy milk}; remove note;}
- What can go wrong?
- Is this better than before?

Too much milk (solution #4)

- Idea: Change order of leaving note and checking note
- Carl: leave noteCarl; if (no noteElsa) {
  if (milk low) {buy milk};} ; remove noteCarl
- Elsa: leave noteElsa; if (no noteCarl) {
  if (milk low) {buy milk};} ; remove noteElsa
- What can go wrong?
- Is this better than before?

Too much milk (solution #5)

- Idea: When both leave note, always give priority to fixed one to buy milk
- Carl: leave noteCarl; while (noteElsa) {do nothing};
  if (milk low) {buy milk}; remove noteCarl
- Elsa: leave noteElsa; if (no noteCarl) {
  if (milk low) {buy milk};} ; remove noteElsa

Simplified instance of Bakery Algorithm