CS162
Operating Systems and Systems Programming
Lecture 4

Synchronization, Atomic operations, Locks, Semaphores

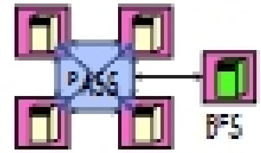
January 31, 2011

Ion Stoica

http://inst.eecs.berkeley.edu/~cs162

Space Shuttle Example

- Original Space Shuttle launch aborted 20 minutes before scheduled launch
- Shuttle has five computers:
  - Four run the "Primary Avionics Software System" (PASS)
    - Asynchronous and real-time
    - Runs all of the control systems
    - Results synchronized and compared every 3 to 4 ms
  - The Fifth computer is the "Backup Flight System" (BFS)
    - stays synchronized in case it is needed
    - Written by completely different team than PASS



- Countdown aborted because BFS disagreed with PASS
  - A 1/67 chance that PASS was out of sync one cycle
  - Bug due to modifications in initialization code of PASS
    - A delayed init request placed into timer queue
    - As a result, timer queue not empty at expected time to force use of hardware clock
  - Bug not found during extensive simulation

Another Concurrent Program Example

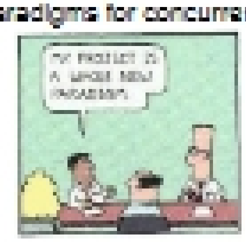
- Two threads, A and B, compete with each other
  - One tries to increment a shared counter
  - The other tries to decrement the counter

Thread A	Thread B
i = 0;	i = 0;
while (i < 10)	while (i > -10)
i = i + 1;	i = i – 1;
printf("A wins!");	printf("B wins!");

- Assume that memory loads and stores are atomic, but incrementing and decrementing are *not* atomic
- Who wins?
- Is it guaranteed that someone wins? Why or why not?
- What it both threads have their own CPU running at same speed? Is it guaranteed that it goes on forever?

Goals for Today

- Synchronization
- Hardware Support for Synchronization
- Higher-level Synchronization Abstractions
  - Semaphores, monitors, and condition variables
- Programming paradigms for concurrent programs



Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne. Many slides generated by Kubiatowicz.

Motivation: "Too much milk"

- Great thing about OS's -- analogy between problems in OS and problems in real life
  - Help you understand real life problems better
  - But, computers are much stupider than people
- Example: People need to coordinate:

Time	Person A	Person B
3:00	Look in fridge. Out of milk	
3:05	Leave for store	
3:10	Arrive at store	Look in fridge. Out of milk
3:15	Buy milk	Leave for store
3:20	Arrive home, put milk away	Arrive at store
3:25		Buy milk
3:30		Arrive home, put milk away

Definitions

- Synchronization: using atomic operations to ensure cooperation between threads
  - For now, only loads and stores are atomic
  - We'll show its hard to build anything useful with only reads and writes
- Mutual Exclusion: ensuring that only one thread does a particular thing at a time
  - One thread excludes the other while doing its task
- Critical Section: piece of code that only one thread can execute at once
  - Critical section is the result of mutual exclusion
  - Critical section and mutual exclusion are two ways of describing the same thing.

More Definitions

- Lock: prevents someone from doing something
  - Lock before entering critical section and before accessing shared data
  - Unlock when leaving, after accessing shared data
  - Wait if locked
    - Important: exec all synchronization involves waiting

- For example: fix the milk problem by putting a key on the refrigerator
  - Lock it and take key if you are going to go buy milk
  - Fixes too much: roommate angry if only wants orange juice

  - Of Course – We don't know how to make a lock yet

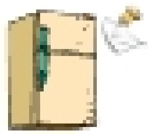
Too Much Milk: Correctness Properties

- Need to be careful about correctness of concurrent programs, since non-deterministic
  - Always write down behavior first
  - Impulse is to start coding first, then when it doesn't work, pull hair out
  - Instead, think first, then code
- What are the correctness properties for the "Too much milk" problem?
  - Never more than one person buys
  - Someone buys if needed
- Restrict ourselves to use only atomic load and store operations as building blocks

Too Much Milk: Solution #1

- Use a note to avoid buying too much milk:
  - Leave a note before buying (kind of "lock")
  - Remove note after buying (kind of "unlock")
  - Don't buy if note (wait)
- Suppose a computer tries this (remember, only memory read/write are atomic):

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



- Result?

Too Much Milk: Solution #1

- Still too much milk but only occasionally!

```
Thread A                        Thread B
if (noMilk)
  if (noNote) {
                                if (noMilk)
                                  if (noNote) {
      leave note;
      buy milk;
      remove note;
    }
}
                                    leave note;
                                    buy milk;
                                    …
```

- Thread can get context switched after checking milk and note but before buying milk!
- Solution makes problem worse since fails intermittently
  - Makes it really hard to debug…
  - Must work despite what the thread dispatcher does!

Too Much Milk: Solution #1½

- Clearly the Note is not quite blocking enough
  - Let's try to fix this by placing note first
- Another try at previous solution:

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

- What happens here?
  - Well, with human, probably nothing bad
  - With computer: no one ever buys milk

Too Much Milk Solution #2

- How about labeled notes?
  - Now we can leave note before checking
- Algorithm looks like this:

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

- Does this work?