Lecture 5: Synchronization: Too Much Milk

5.0 Motivation

	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. Oh no!

5.1 Definitions

Synchronization: using atomic operations to ensure cooperation between threads

Mutual exclusion: ensuring that only one thread does a particular thing at a time. One thread doing it *excludes* the other, and vice versa.

Critical section: piece of code that only one thread can execute at once. Only one thread at a time will get into the section of code.

Lock: prevents someone from doing something.

1) Lock before entering critical section, before accessing shared data
2) unlock when leaving, after done accessing shared data
3) wait if locked

Key idea -- all synchronization involves waiting.

5.2 Too Much Milk: Solution #1

What are the correctness properties for the too much milk problem?

never more than one person buys

someone buys if needed

Restrict ourselves to only use atomic load and store operations as building blocks.

Basic idea of solution #1:

1. Leave a note (kind of like "lock")
2. Remove note (kind of like "unlock")
3. don't buy if note (wait)

Solution #1:

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

Why doesn't this work? Thread can get context switched after checking milk and note, but before buying milk!

Our "solution" makes problem worse -- fails only occassionally. Makes it really hard to debug. Remember, constraint has to be satisfied, independent of what the dispatcher does -- timer can go off, and context switch can happen at any time.

5.3 Too Much Milk Solution #2

How about labelled notes? That way, we can leave the note before checking the milk.

Solution #2:

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

Possible for neither thread to buy milk; context switches at exactly the wrong times can lead each to think the other is going to buy.
Illustrates **starvation**: thread waits forever

5.4 Too Much Milk Solution #3

Solution #3:

```
  Thread A                      Thread B
leave note A                   leave note B
while (note B)// X             if (noNoteA){// Y
   do nothing; if                if (noMilk)
(noMilk)                           buy milk
   buy milk;                     }
remove note A                  remove note B
```

Does this work? Yes. Can guarantee at X and Y that either
  (i) safe for me to buy
  (ii) other will buy, ok to quit

At Y: if noNote A, safe for B to buy (means A hasn't started yet)

    if note A, A is either buying, or waiting for B to quit,
     so ok for B to quit

At X: if nonote B, safe to buy
    if note B, don't know. A hangs around. Either:
     if B buys, done
     if B doesn't buy, A will.

5.5 Too Much Milk Summary

Solution #3 works, but it's really unsatisfactory:

1. really complicated -- even for this simple an example, hard to convince yourself it really works
2. A's code different than B's -- what if lots of threads? Code would have to be slightly different for each thread.
3. While A is waiting, it is consuming CPU time (**busy-waiting**)

There's a better way.
1. Have hardware provide better (higher-level) primitives than atomic load and store. Examples in next lecture.

2. Build even higher-level programming abstractions on this new hardware support. For example, why not use locks as an atomic building block (how we do this in the next lecture):

  Lock::Acquire -- wait until lock is free, then grab it
  Lock::Release -- unlock, waking up a waiter if any

These must be atomic operations -- if two threads are waiting for the lock, and both see it's free, only one grabs it!

With locks, the too much milk problem becomes really easy!

```
    lock->Acquire();
```

```
if (nomilk)
  buy milk;
lock->Release();
```