5 Synchronization: The Too-Much-Milk Problem

Readings for this topic: Siberschatz/Galvin: Chapter 6

Synchronization: using atomic operations to ensure correct operation of cooperating processes.

The "too much milk" problem:

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 Leave store.	Leave for store.
3:20 Arrive home, put milk away.	Arrive at store.
3:25	Leave store.
3:30	Arrive home. *OH, NO!*

What does correct mean? Somebody gets milk, but we don't get too much milk. One of the most important things in synchronization is to figure out what you want to achieve.

Mutual exclusion: Mechanisms that ensure that only one person or process is doing certain things at one time (others are excluded). E.g. only one person goes shopping at a time.

Critical section: A section of code, or collection of operations, in which only one process may be executing at a given time. E.g. shopping. It's a large operation that we want to make "sort of" atomic.

There are many ways to achieve mutual exclusion. Most involve some sort of *locking* mechanism: prevent someone from doing something. For example, before shopping, leave a note on the refrigerator.

Three elements of locking:

1. Must lock before using. *leave note*
2. Must unlock when done. *remove note*
3. Must wait if locked. *don't shop if note*

1st attempt at computerized milk buying:

Processes A & B

```
1   if (NoMilk) {
2       if (NoNote) {
3           Leave Note;
4           Buy Milk;
5           Remove Note;
6       }
7   }
```

- This doesn't always work: A1 A2 B1 B2 B3 B4 B5 B6 ...
- This solution works for people because lines 1-3 are performed atomically: you'll see the other person at the refrigerator and make arrangements. Typically, computers don't both test (look for other person) and set (leave note) at the same time.
- In this case, we haven't eliminated the problem; we've just moved it and made it a little less likely (i.e. a little more insidious). This is typical of first attempts at solutions to synchronization problems.

What happens if we leave the note at the very beginning: does this make everything work?

2nd attempt: change meaning of note. A buys if there's no note, B buys if there is a note. This gets rid of confusion.

Process A

```
1   if (NoNote) {
2       if (NoMilk) {
3           Buy Milk;
4       }
5       Leave Note;
6   }
```

Process B

```
if (Note) {
    if (NoMilk) {
        Buy Milk;
    }
    Remove Note;
}
```

- Does this work?
- How can you tell?
- Ideally, we shouldn't rely on intuitions or informal reasoning when dealing with complex parallel programs: we should be able to *prove* that they behave correctly. Unfortunately, formal verification has only been successful on very small programs (too hard to do). For example, in the above example:
  - A note will be left only by A, and only if there isn't already a note.
  - A note will be removed only by B, and only if there is a note.
  - Thus, there is either one note, or no note.
  - If there is a note, only B will buy milk.
  - If there is not a note, only A will buy milk.
  - Thus, only one process will buy milk.
- Suppose B goes on vacation. A will buy milk once and won't buy any more until B returns. Thus this really doesn't really do what we want; it's unfair, and leads to starvation.
- In other words, can't distinguish "You're buying" from "I'm not buying." Not enough information.

3rd attempt: use 2 notes.

Process A

```
1   Leave NoteA;
2   if (NoNoteB) {
3       if (NoMilk) {
4           Buy Milk;
5       }
6   }
7   Remove NoteA;
```

Process B is the same except interchange NoteA and NoteB.

What can you say about this solution? (Look at the boundary between statements 1 and 2)

- At most one process will buy milk: each process leaves note before it checks.
- If one process goes on vacation after step 7, the other process will still buy milk.
- Suppose both A and B leave notes at exactly the same time: nobody will buy milk (there is still starvation).

Solution is almost correct. We just need a way to decide who will buy milk when both leave notes (somebody has to hang around to make sure that the job gets done).

4th attempt: in case of tie, B will buy milk. Process A stays the same as before.

Process B

```
1   Leave NoteB;
2   while (NoteA) DoNothing;
3   if (NoMilk) {
4       Buy Milk;
5   }
6   Remove NoteB;
```

This solution works. But it still has two disadvantages:

- Asymmetric (and complex) code.
- While B is waiting it is consuming resources (busy-waiting).