CS 537 Notes, Section #5: Synchronization: The Too Much Milk Problem

---

Chapter 6, Sections 6.2 and 6.5, in **Operating Systems Concepts**.

Review idea of atomic operations. Example: counting contest.

```
static int i;
```

Process A

```
i = 0;
while (i < 10) {
    i++;
}
cout << "A wins";
```

Process B

```
i = 0;
while (i > -10) {
    i--;
}
cout << "B wins";
```

- Variable i is shared.
- Reference and assignment are each atomic.
- Will process A or process B win?
- Will they ever finish?
- If one finishes, will the other also finish?
- Does it help A to get a head start?

Synchronization: the use of atomic operations to ensure the 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? 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 is a large operation that we want to make "sort of" atomic.

There are many ways to achieve mutual exclusion, which we will be discussing all of this week. 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.	*do not 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	`}`

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 is no note, B buys if there is a note. This gets rid of confusion.

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

- Does this work?
- How can we tell?
- When dealing with complex parallel programs, we cannot rely on our intuitions or informal reasoning. We need to *prove* that the programs behave correctly. What can we say about the above solution?

Suppose B goes on vacation. A will buy milk once and will not buy any more until B returns. Thus this really does not really do what we want; it is unfair, and leads to starvation.

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 we say about this solution?

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, A will buy milk:

Process B stays the same as before.

	Process A
1	`Leave NoteA;`
2	`if (NoNoteB) {`
3	`    if (NoMilk) {`
4	`        Buy Milk;`
5	`    }`
6	`} else {`
7	`    while (NoteB) {`
8	`        DoNothing;`
9	`    }`
10	`    if (NoMilk) {`
11	`        Buy Milk;`
12	`    }`
13	`}`
14	`Remove NoteA;`

How do we know this is correct?

This solution works. But it still has two disadvantages:

- A may have to wait while B is at the store.
- While A is waiting it is consuming resources (busy-waiting).
