

## CS 414 : Operating Systems

UNIVERSITY OF VIRGINIA  
Department of Computer Science

Spring 2008

### Topic 3: Cooperating Processes and Synchronization

- *Synchronization*: using atomic operations to ensure correct cooperation between processes.
- The “Too Much Milk” problem:

	<b>Person A</b>	<b>Person B</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. <i>OH, NO!</i>

- One of the most important things in synchronization is to figure out what you want to achieve.  
In the given problem: *somebody gets milk, but we don't get too much milk.*
- *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.

- 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  }
```

- Will it work?
- If not, why not?
- Making the problem less likely to occur - that 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.

**Process A**

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

**Process B**

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

- Does this work? How can you tell?
- Ideally, we shouldn't rely on intuitions or informal reasoning when dealing with 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. 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.

- 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?

- 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;
```

- Does this solution work?
- Any disadvantages?