Classical Process Synchronization Problems

Too much Milk Problem
Cigarette Smoker's Problem
Barber Shop Problem
etc…

Strong Writer Priority

- Recall the Readers-Writers problem.
- With the strong writer priority, we would like to allow waiting writers to access the DB before any waiting readers.

Readers:

```
P(m1)
  P(readlock)
    P(m2)
      rcount++
      if (rcount==1)  P(writelock)
    V(m2)
  V(readlock)
V(m1)
Read
P(m2)
  rcount--
  if (rcount==0)  V(writelock)
V(m2)
```

Writers:

```
P(m3)
  wcount++
  if (wcount==1)  P(readlock)
V(m3)
P(writelock)
Write
V(writelock)
P(m3)
  wcount--
  if (wcount==0)  V(readlock)
V(m3)
```

Study this solution carefully!!

Using Locks

A locking mechanism is used to prevent other threads from accessing certain parts of the code. A thread that performs the *Lock-operation* is said to acquire the lock as the owner of it.

→ of course, semaphores are an ideal mechanism for locking sections of code!!

Fundamental Locking Roles:

A thread or process should acquire the lock before entering its CS.

A thread should release the lock (unlock) when leaving the CS.

Unlock-operations can only be performed by the owner of the lock.

Only one thread at a time can be the owner of a lock

A thread must wait if it is unable to acquire the lock.

The too-much-milk problem!

Time	You	Roommate
3:00	Arrive Home	
3:05	Look in fridge; no milk!	
3:10		Arrive Home
3:15		Look in fridge; no milk!
3:20	Leave for grocery	Leave for grocery
3:25	Buy milk	
3:35	Arrive Home; put milk in fridge	
3:45		Buy milk
3:50		Arrive Home; OH NO!

What are the goals?

- What is it that we wish to achieve by synchronizing the two threads (you and your roommate)?
  - Only on person buys milk at a time;
  - Someone always buys milk if needed;
- We will use basic atomic building blocks:
  - Leave a note (set a flag) - *Locking*
  - Remove a note (reset a flag) - *Unlocking*
  - Do not buy milk if there is a note (test the flag) - *must wait*

Solution #1

You:

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

Your Roommate:

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

A naïve solution! This does not work.

WHY?

Solution#2

Thread A

```
Leave NoteA;
   if (NoNoteB) {
     if (NoMilk) {
         Buy Milk;
     }
   }
Remove NoteA;
```

Thread B

```
Leave NoteB;
   if (NoNoteA) {
     if (NoMilk) {
         Buy Milk;
     }
   }
Remove NoteB;
```

What if both, A & B leave a note?

Solution #3

You (Thread A)

```
if (NoNote) {
if (NoMilk) {
        Buy Milk;
}
    Leave Note;
}
```

Roommate (Thread B)

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

Does this work? Why / Why Not??

What if you go on vacation?

A correct (but clumsy) solution

```
Thread A:
Leave NoteA;
   if (NoNoteB){
       if (NoMilk)
            Buy Milk;
    } else
        while (NoteB)
          DoNothing;
        If (NoMilk)
          Buy Milk;
Remove NoteA;
```

```
Thread B:

Leave NoteB;
   if (NoNoteA) {
      if (NoMilk) {
          Buy Milk;
      }
   }
Remove NoteB;
```


A semaphore-based solution

Both Threads - A & B

```
OKToBuyMilk.P();

if (NoMilk) BuyMilk();

OKToBuyMilk.V();
```


The Barbershop problem

- This synchronization problem closely resembles a real-world synchronization problem.
- We consider a barbershop with the following characteristics:
  - 3 barber chairs
  - 3 barbers
  - a waiting area for at most k customers (Fire codes)
  - sofa for 4 customers
  - extra standing room for k – 4.
  - 1 cash register
- Customers will not enter the shop if it is filled to capacity!

Once inside, a customer will either sit on the sofa or stand if the sofa is filled.

When a barber is free, it will serve the customer from the sofa that is waiting the longest.

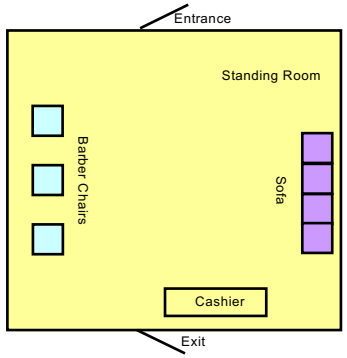
The customer that has been standing the longest will get the seat on the sofa.

When haircut is complete, any barber can accept payment but only 1 customer at a time.

When there are no customers, the barbers are sleeping!

..need for coordination

- Shop and sofa capacity must be controlled and protected
- Barber chair capacity → there must be one available before a customer can sit down.
- Make sure that the customer is in the chair before barber starts cutting hair.
- Hold the customer in the chair until ready.
- Paying and issuing a receipt.
- Coordinating barber and cashier functions.



The Cigarette Smoker Problem

- Consider 3 processes, X, Y, and Z, that supply tobacco, matches, and wrappers as follows:
  - X supplies tobacco and a match
  - Y supplies a match and a wrapper
  - Z supplies a wrapper and tobacco
- Three smoker processes, A, B, C, posses tobacco, matches, and wrappers, respectively.
- However, to smoke, they need all three items.
- Your task, if you choose to accept it, is to write processes A, B, and C.

X, Y, Z, A, B, and C have the following constraints:

only one X, Y, or Z can supply the needed material at a time.

A, B, and C cannot proceed until the missing material is available.

Neither of X, Y, and Z can proceed until the items they supplied have been consumed by the smokers.

...now we are smoking....

- Processes X, Y, and Z are easily written using simple semaphores:

```
Process X
loop
  P(s);
  V(t);
  V(m);
endloop
```

```
Process Y
loop
  P(s);
  V(m);
  V(w);
endloop
```

```
Process Z
loop
  P(s);
  V(w);
  V(t);
endloop
```

So, what is the problem you need to in order to formulate a solution for the smoker processes?

Next, we will be looking at advanced synchronization mechanisms including:

Sequencer and event counts

AND Synchronization (i.e., parallel semaphores)

Monitors

Coordination Languages

- CSP
- Path Expressions