Synchronization Review

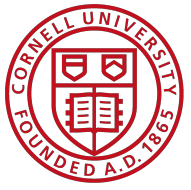
CS 4410, Operating Systems

Fall 2016
Cornell University

Rachit Agarwal
Anne Bracy

See: Ch 5&6 in OSPP textbook

The slides are the product of many rounds of teaching CS 4410 by Professors Sirer, Bracy, Agarwal, George, and Van Renesse.



Synchronization: Topic 1

Synchronization Motivation & Basics

- Race Conditions
- Critical Sections
- Example: Too Much Milk
- Basic Hardware Primitives
- Building a SpinLock

Synchronization: Topic 2

Semaphores

- Definition
- Binary Semaphores
- Counting Semaphores
- Implementing Semaphores
- Classic Synchronization Problems (w/Semaphores)
    - Producer-Consumer (w/ a bounded buffer)
    - Readers/Writers Problems
- Classic Semaphore Mistakes
- Semaphores Considered Harmful

Synchronization: Topic 3

Monitors & Condition Variables

- Definition
- Semantics
- Simple Monitor Example
- vs. Semaphores
- Classic Synchronization Problems (w/Monitors)
    - Bounded Buffer Producer-Consumer
    - Readers/Writers Problems
- Classic Synchronization Mistakes (w/Monitors)

What is a Monitor?

[Hoare 1974]

ADT for shared resources:

1. Shared Private Data
   - the resource
   - accessed only here
2. Procedures
   - to access resource
   - only act on data local to the monitor
3. Synchronization primitives
   - among threads that access the procedures

```
Monitor stack
{
    int top;

    void push(any_t *) {
    }

    any_t *pop() {
    }

    initialization_code() {
    }
}
```

*Monitors can define **Condition Variables***

A mechanism to wait for events

3 operations on Condition Variable `Condition x;`

- **`x.wait()`**: release monitor lock, relinquish processor, sleep until woken up (or wake up on your own), reacquire on return
- **`x.signal()`**: wake at least one process waiting on condition (if there is one). No history associated with signal.
- **`x.broadcast()`**: wake all processes waiting on condition (useful for resource manager)

You **must** hold the monitor lock to call these operations.

Types of Wait Queues

Monitors have two kinds of "wait" queues

- **Entry to the monitor:** has a queue of threads waiting to obtain mutual exclusion & enter
- **Condition variables:** each condition variable has a queue of threads waiting on the associated condition

Monitors in Python

```
class RWlock:
  def __init__(self):
     self.lock = Lock()
     self.canRead = Condition(self.lock)
     self.canWrite = Condition(self.lock)
     self.nReaders = 0
     self.nWriters = 0
     self.nWaitingReaders = 0
     self.nWaitingWriters = 0

  def begin_read(self):
     with self.lock:
        self.nWaitingReaders += 1
        while self.nWriters > 0 or self.nWaitingWriters > 0:
           self.canRead.wait()
        self.nWaitingReaders -= 1
        self.nActiveReaders += 1

  def end_read(self):
    with self.lock:
        self.nReaders -= 1
        if self.nReaders == 0 and self.nWaitingWriters > 0:
                    self.canWrite.notify()
```

Do not forget:

- **One lock for the monitor**
- **CVs initialized with the lock**
- **counters initialized**
- **who waits? who notifies?**
- **post-wait updates**
- **notify vs. notifyAll**

Barbershop Problem

One possible version:

- A barbershop holds up to k clients
- N barbers work on clients
- M clients total want their hair cut
- Each client will have their hair cut by the first barber available

Barbershop Problem

Another possible version:

- Barbershop has an exit door
  - customer cannot leave until door is open
  - barber cannot take on a new client until customer has left & closed the door
- Customer takes a seat only when a barber ready
- Barber cut hair only when customer is seated

More Specs

Need to implement three monitor functions:

getHaircut:

- called by client
- returns when haircut is done

getClient:

- called by barber to serve a customer

letClientLeave:

- called by barber to let a customer out of the barbershop

Implementing the Barbershop

(1) Identify the waits

Customer wait:

- until barber is available
- until barber opens exit door

Barber waits:

- until customer sits in a chair
- until customer leaves

(2) Create condition variables for each

(3) Create counters to trigger the waiting

(4) Create signals for the waits

Implementing the Barbershop

(1) Identify the waits

(2) Create condition variables for each

- `waitForBarber`
- `waitForDoor`
- `waitForClient`
- `waitForClientToLeave`

(3) Create counters to trigger the waiting

(4) Create signals for the waits

Implementing the Barbershop

(1) Identify the waits

(2) Create condition variables for each

(3) Create counters to trigger the waiting

- nSeatedCustomers
- nBarbersAvail
- nDoorsOpened

(4) Create signals for the waits