

CS162 – Section # 3, 02/11/2003

Rodrigo Fonseca

CVS Quick Start Guide for CS162

<http://inst.eecs.berkeley.edu/~cs162/projects/cvs.html>

Super quick:

Add to .login:

```
cvsgroup cs162-gNNN
```

Verify that you have a identity.pub file

(add it to you login script)

Initially

1 group member:

```
cvsgroup cs162-gNNN
cvs init
cvs import nachos groupid start
```

Be sure to run these commands from the 'nachos' directory!

To work

First time:

```
cd nachos
cvs co nachos
cvs commit
```

Later on:

```
cd nachos
cvs update -d
```

If you create files:

```
cd nachos
cvs add filenames...
```

To remove files:

1. remove it locally: `rm filename`
2. Then, remove from CVS: `cvs remove filename; cvs commit`

Notes:

- You can't rename files: remove and then add with new name, but loses history
- Do not add temporary directories: they can't be removed
- It is best that you only add source files, or files that can't be derived from the others

Nachos:

Design document: no code, state assumptions, reasoning, correctness constraints

No busy waiting for solutions!

Synchronization structures

Motivations: threads need to cooperate!

Reading

Silberchatz:

Chapter 7 (up to 7.7) - Synchronization

Chapter 6 – CPU Scheduling (6.5 talks about priority inversion)

Birrell, "[An Introduction to Programming with Threads](http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-035.html)", Jan, 1989

<http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-035.html>

Concepts

- Critical section: code that is to be accessed in mutual exclusion
- Mutual exclusion: only one does something at a time
- Atomic operation: indivisible

Lock, use, Unlock. Wait if locked

High Level Primitives

Semaphores

- Two *atomic* operations:
 - P() and V() (decrement/increment; wait/signal)
- Implementation:
 - Book (Section 7.4.2)
- A non-negative integer **value**
 - Can't be read or written, just initialized

Example:

Car rental has 15 cars. Inits semaphore with 15.

When customer arrives:

```
sem->P();
```

Go for a ride

```
sem->V();
```

Mutual exclusion:

```
sem->Init(1);
```

Threads:

```
sem->P();
```

```
do something;
```

```
sem->V();
```

Trivia:

- Invented in the 60's by Dijkstra (appear incidentally in the appendix of a paper!)
- P stands for *proberen* (test) and is used to decrement
- V stands for *verhogen* (increment) and is used to increment

Problem: semaphores have different uses: mutual exclusion and scheduling constraints

Code hard to write, understand, debug.

Monitors: clearly separates both functions, by using Locks and Condition Variables

Locks:

Implements mutual exclusion

Acquire() and Release()

Acquire() waits if not free; Release() wakes up anyone waiting

Condition Variables

A condition one waits for, and someone else satisfies

Three operations:

- **sleep()** - wait for a condition to become true or an event to happen. Sometimes called wait().
- **wake()** - signal **one** waiter that that a condition has been satisfied or that an event has occurred. Which waiter being awoken is not specified. Sometimes called signal().
- **wakeAll()** - signal **every** waiter that a condition has been satisfied or that an event has occurred. Sometimes called signalAll() or broadcast().

Condition variables must be used within critical sections, and are associated with a lock. This association is called a monitor.

Before calling any operations on the condition variable, you must hold the lock. Conceptually, this means that you know what you're talking about -- if you're holding the lock, you must know something about the condition. When you call wake() or wakeAll(), you must be holding the lock. This indicates that you can and did make the condition true. When you call sleep, you must also hold the lock. This indicates that you checked if the condition was true before waiting for it to be fulfilled. When you actually sleep, the lock is released. When you awake from sleep(), you hold the lock again.

Example: Donuts in the kitchen

John goes to the kitchen:

```
Acquire lock to donut box
while there are no donuts wait on CV //release lock
take a donut
Release donut box lock
```

Fred goes to the kitchen:

```
Acquire lock to donut box
while there are no donuts wait on CV //release lock
take a donut
```

Release donut box lock

Jim goes to the kitchen:

Acquire lock to donut box
put a donut in the box
Call wake on CV
Release donut box lock

Question:

Why do you need the while?

Who will get the donut?

Any thoughts on how to give Jim control of who gets a donut?

Important: wait does the following:

Atomically releases the lock and goes to sleep

After waking up, reacquire the lock