# Lecture 4: Monitors

K. V. S. Prasad
Dept of Computer Science
Chalmers University
10 Sep 2012

# Questions?

- Anything you did not get
- Are we too fast/slow?
- Have you joined the google group? You must, to mail us and get replies
  - Please don't mail us at our personal addresses
- Found a lab partner?
- Haven't yet heard from all course reps

# Plan for today

- Chap 2 (final questions)
- Chap 6 : Semaphores
  - Recap
  - Buffer, infinite and bounded
  - Dining philosophers
- Chap 7: Monitors (get started)

Chap 3 & 4 (skipped for now)

# The standard Concurrency model

1. What world are we living in, or choose to?
   a. Synchronous or asynchronous?
   b. Real-time?
   c. Distributed?
2. We choose an abstraction that
   a. Mimics enough of the real world to be useful
   b. Has nice properties (can build useful and good programs)
   c. Can be implemented correctly, preferably easily

# Obey the rules you make!

1. For almost all of this course, we assume single processor without real-time (so parallelism is only potential).
2. Real life example where it is dangerous to make time assumptions when the system is designed on explicit synchronisation – the train
3. And at least know the rules! (Therac).

# Goals of the course

- covers parallel programming too – but it will not be the focus of this course
- Understanding of a range of programming language constructs for concurrent programming
- Ability to apply these in practice to synchronisation problems in concurrent programming
- Practical knowledge of the programming techniques of modern concurrent programming languages

# Semantics

- What do you want the system to do?
- How do you know it does it?
- How do you even say these things?
  - Various kinds of logic
- Build the right system (Validate the spec)
- Build it right (verify that system meets spec)

# Semaphore recap

- Avoid busy waiting
- Look good for n-proc CS problem
- But for the producer-consumer problem
  - The correctness of each proc
    - Depends on the correctness of the other
  - Not modular
- Monitors modularise synchronisation for shared memory

# Correct?

- Look at state diagram (p 112, s 6.4)
  - Mutex, because we don't have a state (p2, q2, ..)
  - No deadlock
    - Of a set of waiting (or blocked) procs, one gets in
    - Simpler definition of deadlock now
      - Both blocked, no hope of release
  - No starvation, with fair scheduler
    - A wait will be executed
    - A blocked process will be released

# Producer - consumer

- Yet another meaning of ”synchronous”
  - Buffer of 0 size
- Buffers can only even out transient delays
  - Average speed must be same for both
- Infinite buffer first. Means
  - Producer never waits
  - Only one semaphore needed
  - Need partial state diagram
  - Like mergesort, but signal in a loop
- See algs 6.6 and 6.7

# Infinite buffer is correct

- Invariant
  - #sem = #buffer
    - 0 initially
    - Incremented by append-signal
      - Need more detail if this is not atomic
    - Decremented by wait-take
- So cons cannot take from empty buffer
- Only cons waits – so no deadlock or starvation, since prod will always signal

# Bounded buffer

- See alg 6.8 (p 119, s 6.12)
  - Two semaphores
    - Cons waits if buffer empty
    - Prod waits if buffer full
  - Each proc needs the other to release ”its” sem
    - Different from CS problem
  - ”Split semaphores”
  - Invariant
    - notEmpty + notFull = initially empty places

# Dining Philosophers

- Obvious solution deadlocks (alg 6.10)
- Break by limiting 4 phils at table (6.11)
- Or by asymmetry (6.12)