CS 350 Notes

Matthew Visser

Sep 27, 2011

1 Semaphores

1.1 The Producer-Consumer Problem

Producers add items (to a list) and consumers consume them.

Need to be careful how you apply semaphores, if you reverse an order or don't think about them, you could end up with thread blocking.

You also don't want consumers continually creating items. You can limit this with a semaphore in the other direction.

There are many types of synchronization problems:

- Dining philosopher
- Sleeping barber
- Cigarette smoker
- Readers and Writers

2 Locks

Locks are used to enforce mutual exclusion. You can use them in OS/161 by calling `lock_aquire()` and `lock_release()`.

The implementation needs to enforce that the thread that acquires the lock is the thread that releases the lock.

3 Condition Variables

A thread can call three operations

wait: Causes a thread to wait for a condition. Releases the lock it has on the critical section. The lock is re-acquired before returning from the wait procedure.

signal If threads are blocked, one of them will be unblocked.

broadcast If threads are blocked, all of them are unblocked.

There are two semantics of using condition variables:

Mesa Semantics: Signalling thread stays in its CS. Waiting thread must wait for signalling thread and possibly other threads after unblocked.

Hoare Semantics: Signalling thread leaves CS and waiting thread atomically enters its CS.

Semaphores accumulate calls to `V()`. Condition variables do not accumulate, they either broadcast or signal.

4 Monitors

You can simulate having monitors by using a design pattern and code conventions, listed on slide 30 of `synch.pdf`.

5 Dead Locks

Dead locks can occur when you have multi-threaded code. This type of problem happens when two threads are permanently waiting for each other. This usually happens when a thread holds one resource, and asks for another.

Desirable properties in multi-threaded programs are:

- Safety – threads are properly synchronized.
- Liveness – threads all make progress (eventually). There are three violations to this:
  - Starvation
  - Dead Lock
  - Live Lock

Having a cycle in resource allocation is a necessary condition for a dead lock, but it isn't sufficient.

5.1 Dead Lock Prevention

No Hold and Wait: Only request resources one at a time, if requesting multiple, request at once.

Preemption: Take resources away from a thread and give them to another.

Resource Ordering: Assign each resource a number. Always request them in increasing order, *i.e.*, always request lock A before lock B, no matter what thread or function.

5.2 Dead Lock Recovery

Need to detect there is a deadlock, then terminate one thread causing the lock.