Lecture 5: Coordination 1

Adam Hawley

January 29, 2019

1 Interacting Processes

Tasks need to share data. They do this by coordinating, here are some examples of tasks which require coordination:

- One at a time through a critical section
- A starts X after B finishes Y
- Replicated tasks need to combine/compare results
- Work needs to be allocated to a manager

There are two approaches to interacting processes: **shared variables** and **message passing**.

1.1 Mutual Exlusion — Mutex

Critical sections are code that must not be executed by more than one task at a time. Unfortunately implementations of mutual exclusion are often complex and error prone. They also do not easily generalise to n tasks nor do they easilly generalise to more complex problems.

1.2 Levels of Support

Simple primitive:

- Semaphores — simple processes for guaranteeing mutual exclusion.
- Mutexes (normally provided by the runtime environment/OS so not discussed in detail).

Control structures:

- Monitors which are normally provided by the language.

1.2.1 Semaphores

A semaphore is a non-negative integer together with two primitives: `wait` and `signal`. On creation, a semaphore is initialised to 1 (in the simplest case).

- `signal` — Atomically incrememnts a semaphore.
- `wait` — If the semaphore has a value greater than zero, decrements the value by 1. If the semaphore is equal to 0 then the executing task **blocks**

Blocking is when a task is not runnable. Tasks are unblocked when its semaphore becomes > 0. If multiple tasks are blocked on a semaphore, `signal(sem)` will unblock one task chosen non-deterministically.