COS 226 : Concurrent Systems

Properties of Concurrent Systems

- **Liveness** - Some property of the system that ensures that something good will happen (eventually).
- **Safety** - Some property of the system that ensures that something bad will *not* happen

Mutual Exclusion

This is the concept that no two threads may be executing their critical sections at any given instance.

If this property is not held up in certain places, it is cause for irregularity and and poor program control.

Starvation

This is a term given to the situation where a **subset of the running threads** are **prevented from receiving processor time**, because another subset of threads is repeatedly occupying the processor.

In this situation, only a select few of the threads *make progress*.

Deadlock

This is the name given to the situation where **all threads are waiting on other threads** before they execute. Hence all the threads are waiting for a condition that cannot be met.

In this case **no threads make any progress**.

The Producer-Consumer Problem

This is a problem that arises when we have two threads that are **mutually depended on the actions of the other** to run.

We do not want the Producer to produce when the buffer is full.

We do not want the Consumer to consume when the buffer is empty.

The Readers-Writers Problem

This is a problem where we have some shared memory that allows communication between two or more threads. It is possibe that one thread may read from the shared memory as it is being modified by another thread. This is likely to cause miscommunication and unanticipated consequences of the system.

Amdahl's Law

Amdahl's Law is a mathematical equation that approximates the speedup of concurrent programs.

Based on the level **parallism** and **number of processors**.

It is unlikely that we can receive more than a 7 times performance increase from making some program concurrent*.

*With reasonable exceptions.

Locks

A lock (or 'mutex') is a synchronisation mechanism for enforcing limits on access to a resource, in an enironment where there are many threads of execution. A good lock:

- **Enforce mutual exclusion**.
- **Deadlock-free**.
- **Starvation-free**.

Filter Lock

We have n-1 levels that threads need to traverse before they can reach the last level (and be processed).

Lamport's Bakery Algorithm

This lock is based on the concept of taking a ticket in a shop. You will:

- Receive a number, greater than all numbers previous.
- Wait until all numbers lower than you have been called
- Get processed by the shop owner

But wait!!!

It is very possible that we will have the case where two or more threads receive the **same number**.

In this situation we would use the thread id to get the relative positions of threads (greater id's waiter for smaller id's).

This means it is effectively "elders first".