synchronisation

#paper-note #concurrency

Coordination in Concurrent Programming

There are extra design considerations when programming concurrent programs.

Considerations

Partitioning
Partitioning is deciding how a program should be split into chunks for different threads.

Placement
Placement is deciding which threads to execute on which hardware.

Communication
Communication is deciding what data should be shared between threads, when it should be shared, and how it should be shared.

Synchronisation
Synchronisation is the means by which threads are protected from each others interference.

Often this is done by mutexes (mutual excluders) placed around critical sections/shared information.

Matters of Life and Death
If synchronisation is done poorly then some threads can stop working for the following reasons:

- starvation
  - one process never gets access because of other processes always using it
- livelock
  - starvation by two processes trying not to starve each other
- priority inversion
  - lower priority processes starving higher priority ones
- deadlock
  - multiple mutexes causing each other to wait on each other
  - all parties of a group waiting for ever for another party to take action

Synchronisation Means

Synchronisation means are techniques for dealing with problems that arise from multithreading.

Critical Sections
A critical section is a part of code that accesses shared resources or would otherwise cause race conditions if multiple threads accessed it concurrently

Semaphores
Semaphores are variables that track how many threads are accessing a shared resource or critical section at a time.

Mutexes
A mutex is a binary semaphore.

Race Condition Rules
A solution for doling out access to avoid race conditions must follow these rules:

- only let one process in a critical section at once
- vary the process that is let into critical sections
- don't starve any processes from entering critical sections