## Synchronisation

1. 1. In today’s modern systems processes are optimized for special tasks. This means that a task is divided up and given to the process that can handle the task best.   
      A simple example is ls – ws process.
   2. In this example ls will send information through a channel/pipe. On the other side the wc-process is waiting for data.
   3. One of the problems that occurs is that the processes have a conflict when accessing the dataspace at the same time. Therefor we need to synchronize the data access.
2. A critical region

* A code segment that accesses shared variables (or other shared resources) and that has to be executed as an atomic action.
* A process can be interrupted in a critical region by a process with higher priority and don’t need to run in a CR.

1. Busy wait vs blocked

* In busy wait the process is essentially consistently running, waiting for an operation to successfully finish. This is ineffective as the waiting thread is constantly loops/busy-waits, using processing power without useful progress. It cant also delay the scheduling of other threads.
* In blocked the process is given a lock that the process uses when a process is waiting for data is can be put on the wait list, and other processes can enter the CR.

1. Race-condition

* A race condition occurs when two or more threads can access shared data and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data. A simple example is feeding the dog. You and your wife both come home from work and feed the dog. Since you don’t tell each other that the dog is feed the dog gets twice as much food and becomes fat.

1. Spin-lock

* A spinlock is a lock which causes a thread trying to acquire it to simply wait in a loop while repeatedly checking if the lock is available. Since the thread remains active but is not performing a useful task, the use of such a lock is a kind of busy waiting. However, for locks that are only held for short periods spinlocks make sense, therefor it is often used in the OS kernel.

1. List

* Deadlock, pooling
* The MCS lock algorithm makes each process wait for their node in the queue to be unlocked. This means each processor will spin on a local variable instead of a variable shared with other processors.
* The RCU algorithm works by readers being able to read from shared data structure even as it is being updated.

1. Deadlocks
2. Deadlock vs resource starvation

* In starvation, a thread fails to make progress for an indefinite period of time.
* Deadlock is a form of starvation but with the stronger condition: a group of threads forms a cycle where none of the threads make progress because each thread is waiting for some other thread in the cycle to take action.

1. Deadlock conditions

* There are four conditions for a deadlock to occur, bounded resources, no preemption, wait while holding and circular waiting.

1. Deadlock detection

* The OS runs an algorithm to check for cycle in the Resource Allocation Graph. If the OS detects a circle, then it is for sure a deadlock. However if there are multiple instances of resources a cycle is a necessary but not sufficient condition for deadlock.

1. Scheduling
2. Uniprocessor scheduling
   1. FIFO
   * FIFO is useful where tasks are roughly equal to size as it will not check how big each task is before queuing them. As they will take roughly the same amount of time they will have the most optimized response time.
   1. MFQ
   * Multi-level feedback queue is designed to pick the best from all the older schedulers and make a better one. Some of its goals is responsiveness, that is has from SJF, where all the shorter tasks have priority in the queue. Low overhead means using less time scheduling the queue, just like in FIFO. Starvation-freedom, where all tasks should make progress, like in round-robin. MFQ has the advantage of low scheduling overhead, but the disadvantage of being inflexible.
3. Multi-core scheduling
   1. Using MFQ on a multi-core system will be ineffective because of three reasons:
      1. Contention for scheduler spinlock.
      2. Cache slowdown due to ready list data structure pinging from one CPU to another.
      3. Limited cache reuse – A thread’s data from last execution is in old cache.
   2. Work stealing is a scheduler where each processor has a queue of work items, tasks, threads. If a processor has no more tasks in its queue, it steals tasks from other processors.