Synchronization Primitives

- **Process Synchronization** means sharing resources with processes such that chances of inconsistent data is minimized with concurrent access to shared data.
  - **Critical section**: is part of the program that accesses shared resources.
  - **Race condition** is prevented by synchronization. This is when several processes access and manipulate shared data at the same time.
  - **Deadlock**: at least two tasks is holding a lock that the other task holds. If nothing is done they will wait forever
- **Atmoic operation** is when an operation completes in its entirety without interruption.
- **Mutual exclusion** is a mechanism to prevent data inconsistency. One one process is doing something at a certain time
  - Also called mutex
- **POSIX**: standard for maintaining compatibality between operating systems

- **Locks**
  - primitive, minimal semantics
  - provide mutual exclusion
  - operations
    - *acquire*(lock);
    - *release*(lock);
  - downsides: can cause deadlock if not careful
- **Semaphores**
  - easy to understand, hard to code
  - generalizes locks with an integer count variable and a thread queue
  - if integer count is negative then threads wait in queue till signalled
  - operations
    - *wait*: down, decrement, P(proberen)
      - block thread till semaphore is free and decrement variable
    - *signal*: up, increment, V(verhogen)
      - increment variable and unblock waiting thread
- **Condition Variables**
  - allow threads to synchronize based on the value of the data
  - useful for implementing monitors
- **Monitors**
  - high level, ideally language supported
  - abstraction that encapsulates shared data and operations in such a way that only one process can be excuting in the monitor
  - only one process at a time can be active in the monitor. Local data accessed only by monitor's procedures. Process enters monitor by invoking one of its procedures. Other processes that attempt to enter monitor are blocked.
  - operations
    - *wait*(): suspend the invoking process and release the lock

        - *signal*(): resume exactly one suspended process which was waiting for its condition variable (if any)
        - *broadcast*(): resume all suspended process which was waiting for its condition variable (if any)
    - Types of monitors (who goes first)
        - **Hoare monitor**: waiter first, switches from caller to waiting thread. Easier to reason with, but hard to implement
        - **Mesa monitor**: signaler first, signler continues while waiter placed on ready queue. Easier to implement, and support additional operations like broadcast
- Thread cycle
    - ready <--> running -> blocked -> ready