Locks:

- a **lock** or **mutex** (from mutual exclusion) is a synchronization mechanism for enforcing limits on access to a resource in an environment where there are many threads of execution. A lock is designed to enforce a mutual exclusion concurrency control policy.
- One way to implement critical sections is to "lock the door" on the way in, and unlock it again on the way out
- Programmers annotate source code with locks, putting them around critical sections, and thus ensure that any such critical section executes as if it were a single atomic instruction.
- Say for example, that there are two threads executing a command, they cannot execute the command at the same time. This is referred to as the critical section. There are 3 sections when it comes executing a thread. These sections are entry, critical, and exit. During the entry one thread can be entered, however it cannot further to the critical section until the OS signals the thread that it is safe to proceed to the critical section. This is where a lock.acquire comes in handy. Once the signal is given, the lock is than released in the critical section.
- A lock is an object in memory providing two operations.
  - acquire(): before entering the critical section
  - release(): after leaving a critical section

Deadlocks:

- a **deadlock** is a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does. This a classic problem in the dining philosopher scenario. Where if all five philosophers pick up the left fork in which case they will have the wait for the right fork to drop, but the right fork won't drop because the signal to drop the fork from the philosophers hasn't gone through. Hence, the philosophers will eventually die of starvation.
- **Mutual Exclusion** - At least one resource must be held in a non-sharable mode; If any other process requests this resource, then that process must wait for the resource to be released. In other words, make sure that one of the process is in the critical section and is executing, therefore there is little confusion between the threads.
- Two other algorithms are Wait/Die and Wound/Wait, each of which uses a symmetry-breaking technique. In both these algorithms there exists an older process (O) and a younger process (Y). Process age can be determined by a time stamp at process creation time. Smaller timestamps are older processes, while larger timestamps represent younger processes. Hence if the waiting occurs for the a long time between the threads than one of them will die, freeing up the other one to enter the critical section.

References:

- https://cseweb.ucsd.edu/classes/fa05/cse120/lectures/120-l5.pdf
- https://en.wikipedia.org/wiki/Deadlock