deadlock: different order to acquire lock or return without release lock
livelock: two threads tries to avoid deadlock, and blocks each other with simultaneous move
mutex lock: don't sleep protect resources from multiple editing
semaphore: sleep, have a task list
binary_semaphore: more synchronization (notify the taker by the giver) than mutual exclusion
counting_semaphore: limit the number of threads to access a resource
reader-writer semaphores: multiple read or one write
monitor: monitors several threads, only run one thread at one time
A monitor has four components as shown below: initialization, private data, monitor procedures, and monitor entry queue. The initialization component contains the code that is used exactly once when the monitor is created, The private data section contains all private data, including private procedures, that can only be used within the monitor. Thus, these private items are not visible from outside of the monitor. The monitor procedures are procedures that can be called from outside of the monitor. The monitor entry queue contains all threads that called monitor procedures but have not been granted permissions. Seems like a mini OS.
conditional variable: The condition_variable class is a synchronization primitive that can be used to block a thread, or multiple threads at the same time, until another thread both modifies a shared variable (the condition), and notifies the condition_variable.
Process VS Threads: The typical difference is that threads (of the same process) run in a shared memory space, while processes run in separate memory spaces.
threads share all segments except the stack. Threads have independent call stacks (and kernel stacks), however the memory in other thread stacks is still accessible and in theory you could hold a pointer to memory in some other thread's local stack frame (though you probably should find a better place to put that memory!).
Scheduling:
First in, first out
Earliest deadline first
Shortest remaining time
Fixed priority pre-emptive scheduling
Round-robin scheduling