Threads and Locks for Interviews

Threads

- In java, threads are implemented in two ways:
    - Implementing runnable
    - Extending threads
- Runnable is preferred over extending thread because:
    - A java function can only inherit one thing and hence using runnable allows it to extend to another class.
    - Inheriting the whole thread class would be very heavy and hence implementing runnable would work well.

Synchronization

- Synchronization blocks restrict multiple threads to execute some function simultaneously.
- Synchronization allows two threads to allow a method once at a time.

Locks

- This is same as synchronization but it works in a way that a thread first has to acquire a lock and threads have to wait. After that, the thread is unlocked.
- Locks can be used from any part to any part of the code but synchronization is only for methods.

DeadLock

- A deadlock is a situation where a thread is waiting for a lock that another thread holds and the second thread is waiting for a lock that the first thread holds.
- This can be with several threads as well.
- Since each of them are waiting for the other's lock, they are stuck forever.
- For deadlock, the following conditions have to be met.
    - Only one process can access a resource at a given time.
    - Process already holding a lock can ask for other locks without giving up on the first locks.
    - One process cannot remove other processes resource
    - Two or more processes form a circular chain where each process holds lock for another process.
- Most deadlock prevention methods rely on getting rid of the last condition.
- In deadlock, all processes are in a waiting state.
- To prevent deadlock:
    - lock timeout should be used,
    - You can do lock ordering in which the lock has to be acquired one after another

Livelock

- In livelock, two threads are busy responding to one another and do not move forward. If thread 2 is doing some action from response of thread 1 but thread 1 is doing that action due to the response from thread 2 then they are stuck.
- Thread 1 does action in response to thread 2.
- Thread 2 does action in response to thread 1.
- If you are going in a corridor and someone is going in the front you keep moving on the same side and never able to pass.
- It can always be avoided by instruction timeout.

Mutex

- A mutex (mutual exclusion object) is a program object that is created so that multiple program thread can take turns sharing the same resource, such as access to a file.
- Typically, when a program is started, it creates a mutex for a given resource at the beginning by requesting it from the system and the system returns a unique name or ID for it.
- After that, any thread needing the resource must use the mutex to lock the resource from other threads while it is using the resource. If the mutex is already locked, a thread needing the resource is typically queued by the system and then given control when the mutex becomes unlocked (when once more, the mutex is locked during the new thread's use of the resource).

Semaphores

- A semaphore, in its most basic form, is a protected integer variable that can facilitate and restrict access to shared resources in a multi-processing environment.
- The two most common kinds of semaphores are counting semaphores and binary semaphores.
- Counting semaphores represent multiple resources, while binary semaphores, as the name implies, represents two possible states (generally 0 or 1; locked or unlocked).
- Semaphores can be looked at as a representation of a limited number of resources, like seating capacity at a restaurant. If a restaurant has a capacity of 50 people and nobody is there, the semaphore would be initialized to 50. As each person arrives at the restaurant, they cause the seating capacity to decrease, so the semaphore in turn is decremented. When the maximum capacity is reached, the semaphore will be at zero, and nobody else will be able to enter the restaurant.
- Semaphores are more like an event object like tripping of a wire or something.
- Mutex on the other hand is used for queueing threads.
- Semaphores can also be used for signalling

Process and Threads

- A process cannot use each other's variable unless something like pipelining or a fifo is used.
- Thread can use all the variables in a process.
- Threads run inside a process.

Context Switching

- In computing, a context switch is the process of storing and restoring the state (more specifically, the execution context) of a process or thread so that execution can be resumed from the same point at a later time.
- This enables multiple processes to share a single CPU and is an essential feature of a multitasking operating system.
- Context switches can occur only in kernel mode. Kernel mode is a privileged mode of the CPU in which only the kernel runs and which provides access to all memory locations and all other system resources.

Stack and Heap

- Local variables and methods are stored in a stack.
- Static has static allocation
- Heap has dynamic allocation
- Objects and all are stored in heap.
- Stakc is when we know how big the data is.