Concurrency

General:

- concurrency refers to the general concept of a system with multiple, simultaneous activities
- parallelism refers to the use of concurrency to make a system run faster

Resources:

- each thread has its own stack
- threads in a process share heap

Approaches:

- hyperthreading (thread level concurrency)
  - multiple copies of some of the CPU hardware (program counters,register files) while having only single copies of other parts (units that perform floating-point arithmetic)
  - decides which thread to run on a cycle by cycle basis
- instruction-level parallelism: modern processors can execute multiple instructions at one time
- application level concurrency
  - Benefits
    - overlap useful work with I/O requests
    - separate concurrent logic flow for interactions with humans (resize window)
    - reducing latency by defering work (in Dynamic Storage allocator defer coalescing to concurrent flow at lower priority in CPU spare time)
    - separate logic flows for each client
    - partitioned applications with concurrent flows run better on multiprocessor
  - Methods
    - processes
      - each logic control flow scheduled and maintained by kernel
      - separate virtual address spaces
      - need explicit interprocess communication (IPC) mechanism to communicate
    - I/O multiplexing
      - applications explicitly schedule their own logical flows in the context of a single process

        - logical flows are modeled as state machines that the main program explicitly transitions from state to state as a result of data arriving on file descriptors
        - all flows share the same address space
    * threads
        - run in the context of a single process and are scheduled by the kernel
        - hybrid of the other two approaches
        - scheduled by the kernel like process flows
        - sharing the same virtual address space like I/O multiplexing flows

---

Tools for sharing:

- lock: allows only one thread to enter the part that's locked and the lock is not shared with any other processes
- Mutex Used to provide mutual exclusion
  - ensures at most one process can do something (like execute a section of code, or access a variable) at a time
  - A famous analogy is the bathroom key in a Starbucks
    * only one person can acquire it, therefore only that one person may enter and use the bathroom
    * Everybody else who wants to use the bathroom has to wait till the key is available again
- Monitor: object designed to be accessed from multiple threads
  - member functions or methods of a monitor object will enforce mutual exclusion, so only one thread may be performing any action on the object at a given time
  - if one thread is currently executing a member function of the object then any other thread that tries to call a member function of that object will have to wait until the first has finished
  - no part of the instance can be touched by more than one process at a time
  - A monitor is like a public toilet
    * Only one person can enter at a time
    * They lock the door to prevent anyone else coming in, do their stuff, and then unlock it when they leave
- Semaphore: lower-level object
  - P() and V()functions
  - You might well use a semaphore to implement a monitor
  - A semaphore essentially is just a counter
    * When the counter is positive, if a thread tries to acquire the semaphore then it is allowed, and the counter is decremented
    * When a thread is done then it releases the semaphore, and increments the counter

  - Semaphores are typically used as a signaling mechanism between processes
  - A semaphore is like a bike hire place
    * They have a certain number of bikes
    * If you try and hire a bike and they have one free then you can take it, otherwise you must wait
    * When someone returns their bike then someone else can take it
    * If you have a bike then you can give it to someone else to return, the bike hire place doesn't care who returns it, as long as they get their bike back

---

Issues:

- Deadlock is a condition in which a task waits indefinitely for conditions that can never be satisfied
  - task claims exclusive control over shared resources
  - task holds resources while waiting for other resources to be released tasks cannot be forced to relinguish resources a circular waiting condition exists
- Livelock conditions can arise when two or more tasks depend on and use the some resource causing a circular dependency condition where those tasks continue running forever
  - this blocks all lower priority level tasks from running (these lower priority tasks experience a condition called starvation)
  - A real world example of livelock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time

---

Implementing in Java:

- extend Thread
  - can't extend any more classes
  - write run() function with what ever you want thread to do
    * thread.start() starts new thread (calls run within start)
    * thread.run() will run
- implement runnable
  - can implement more interfaces
  - write run() function with what ever you want thread to do and create thread with runnable in constructor

sources:

- Computer Systems: A Programmers Perspective
- https://www.quora.com/Semaphore-vs-mutex-vs-monitor-What-are-the-differences
- http://stackoverflow.com/questions/7335950/semaphore-vs-monitors-whats-the-difference
- http://stackoverflow.com/questions/2332765/lock-mutex-semaphore-whats-the-difference
- http://javarevisited.blogspot.com/2014/07/top-50-java-multithreading-interview-questions-html