Race Condition

T/F Concurrency

- The programmer should not have to care/know whether concurrency will be true or false
- a multi-threaded program should reach higher interactivity and performance with True and/or False concurrency
- Main Pitfall of concurrency
  High-level programming languages (e.g., anything but assembly, and even not all assembly languages) hide the complexity of operations performed at the CPU level

Race Condition

- an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time
- a bug that leads the program to gives unpredictably incorrect results
- Can happen with F or T concurrency
- **Lost Update**
  when a thread does "x++" and another does "x–" three things can happen
  - Both updates go through, the x is unchanged
  - "x++" update is lost, and the value of x is decremented only
  - "x–" update is lost, and the value of x is incremented only

Critical Section

- region of code in which only one thread can be at a time
- For correctness only one thread can execute the code in a critical section at a time
- does not have to be a contiguous section of code
- critical section corresponds to section(s) of code
- Three Requirements to Execute Critical Sections
  1. Mutual Exclusion
     If a thread is executing in the critical section, then no other thread can be executing in it
  2. Progress
     If a thread wants to enter into a critical section, it will enter it at some point in the future
  3. Bounded Waiting
     Once a thread has declared intent to enter the critical section, there should be a bound on the number of threads that can enter the critical section before it
- Common Misconception for Critical Sections
  - Doesn't correspond to the data but rather the sections of code
  - say "we need to protect variable x against race conditions" it means "we need to look at the entire code, see where x is modified, and put all those places in the SAME critical section"
- Critical Sections should be short as possible
  Long critical sections: only one thread can do work for a while, so we have reduced parallelism

Locks

- Spin Lock
  The thread constantly checks whether the lock is available in a while loop
  Spinlocks are very useful for (short) critical sections

- Blocking Lock
  The thread asks the OS to be put in the Waiting/Blocked state and the OS will make the thread Ready whenever the lock has been released by another thread

Deadlock

- another common bug that can happen in concurrent programs
- System with Resources and Processes
  - Resources
  - Processes
    each process can Request a resource of a given type and block/wait until one resource instance of that type becomes available, use/release a resource
- deadlock state happens if every Process is waiting for a resource instance is being held by another process
- Three Condition For Deadlock
  1. Mutual Exclusion
     At least one resource is non-shareable: at most one process at a time can use it
  2. No Preemption
     Resources cannot be forcibly removed from processes that are holding them
  3. Circular Wait
     there exists a set of waiting processes such that a Process is waiting for a resource held by other resources
- if it meets all three conditions then it may occur

Resource Allocation/ Request Graph

- Describing the system can be done precisely and easily with a system resource-allocation-request graph
- set of vertices is made of set of processes and resources types
- set of directed edge
  - request edges
  - assignment edges
- **Theorem**
  If the graph contains no (directed) cycle, then there is no deadlock
  The converse does not guarantee to be true
- If the resource-allocation-request graph contains no (directed) cycle, then there is no deadlock in the system
- The existence of a cycle is a necessary and sufficient condition for the existence of a deadlock
- Deadlock Prevention
  1. Prevention
  2. Avoidance
  3. Detection/Recovery

  OSes do nothing
- Getting Rid of Mutual Exclusion
  In general we cannot design a system in which we don't have some type of mutual exclusion on some types of resources
- Getting rid of No Preemption
  cannot be done in general as the processes may be in the middle of doing something that leaves an inconsistent state