OS Semaphores and Monitors

1. **Deadlocks occur if, and only if, all of the following four conditions are met:**

   1) Mutual Exclusion: Resources are held by processes and cannot be shared
   2) Hold and Wait: At least one process is waiting on a resource while holding a different one
   3) No Resource Preemption: A resource cannot be forcibly removed from the process holding it
   4) Circular Wait:
   • A set of processes P = {P0, P1, ..., Pn} must exist so that:
   • For each i so that 0 ≤ i < n, Pi is waiting on a resource that Pi+1 holds
   • Pn is waiting on a resource that P0 holds

2. **Describe how Windows Synchronization works**

   1) Uses simple spinlocks and mutex locks in the kernel
   2) Adapts spinlocks to interrupt-disabling for single-processor systems
   3) For thread synchronization, Windows provides dispatcher objects with: Mutex locks, Semaphores, Timers, Conditions (called Events)
   4) Lock-guarded shared memory is provided
   5) Critical Section Objects provide mutexes at the user level that only allocate kernel mutexes if they have to

3. **Describe in layman terms the four ways of dealing with deadlocks**

   1) Prevent: Attempt to prevent deadlocks by eliminating one of the four conditions that allows them to occur
   2) Avoid: Attempt to avoid deadlocks by monitoring
   resource allocation with an algorithm
   3) Recover: Attempt to recover from deadlocks by detecting them when they occur and taking action
   4) Ignore: ignore deadlocks and hope they don't happen
   very often

4. **Describe Solaris Synchronization.**

   1) Solaris provides semaphores and conditions as we've
   already described
   2) Adaptive mutexes shift themselves from spinlocks to
   yielding if the kernel determines they're going to have to
   wait any real length of time for the lock
   3) Turnstiles are Solaris' method of queuing for adaptive
   mutexes. Turnstiles are dynamically allocated queues that implement priority inheritance
   4) Every synchronization mechanism available to the kernel is available to the user except priority inheritance.

5. **Except for database systems, how do most modern operating systems handle deadlocks?**

   They just ignore them. Proper deadlock handling is hard and they don't happen very often.

6. **How to avoid spin waits with mutex locks?**

   Turn the mutexes into queues, and the queuing itself is a critical section

7. **Monitors also provides condition variables. Describe them.**

   The conditions are essentially queued semaphores with one important difference in behavior:
   1) A condition cannot be "free" in the sense a lock can
   2) Waiting on a condition always waits
   3) Signaling a condition can only free a process already waiting on it

8. **Name the three problems he mentioned with Disabling Interrupts**

   1) Performance: If the timer is controlled by interrupts, it's going to mess them up
   2) Potential unpredictable consequences: Disabling fundamental bits of the operating system any more often than absolutely necessary is not a good idea
   3) Absolute non-starter on multiprocessor systems

9.	**What are the three fundamental ways of dealing with critical sections?**	1) Disabling interrupts, which will reliably prevent interruption but will significantly impact performance 2) Mutual exclusion locks, aka mutexes or binary semaphores, which provide explicit synchronization functionality at the statement level 3) Monitors, which provide implicit synchronization functionality at the object level
10.	**What are the two types of semaphores?**	Can have counting (unrestricted range) and binary (0 or 1) semaphores. Binary is like locked/unlocked, whereas counting is like a librarian keeping track of how many student rooms (resources) are available for the students (processes).
11.	**What are two standard operations of semaphores?**	wait() and signal()
12.	**What is a critical section?**	A critical section is a segment of code in a cooperating process that: 1) Modifies shared information, and either 2) Can destructively interfere with other processes, or... 3) ...be destructively interfered with by other processes
13.	**What is a deadlock?**	Two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes (most OSes do not prevent or deal with deadlocks)
14.	**What is a monitor?**	A monitor is an abstract data type, i.e., an object, with one extra property: Only one function/process of a monitor can execute at the same time.
15.	**What is a race condition?**	When several processes access and manipulate the same data concurrently
16.	**What is a semaphore?**	A semaphore is like a record of how many units of a particular resource are available, coupled with operations to safely (i.e., without race conditions) adjust that record as units are required or become free, and, if necessary, wait until a unit of the resource becomes available.
17.	**What is Test-and-Set?**	A shared boolean variable lock, implemented atomically by the processor.
18.	**What two scenarios may deadlocks also cause?**	Can cause starvation and priority inversion (lower priority process holds lock needed by higher-priority process)