Summary Outline on the Critical Section Problem

1. Arises when two or more processes/threads share a variable that they write to.
2. Key features of a solution:
   - 2.1. Ensures mutual exclusion
   - 2.2. Progress: Solution does not give rise to deadlock. (Note that deadlock is still possible if the processes behave badly - only required that the solution itself does not give rise to deadlock if processes behave well.)
   - 2.3. BoundedWaiting/Fairness: Solution does not create a situation where a process may starve: (Note that starvation is still possible if the processes behave badly - only required that the solution itself does not give rise to starvation if processes behave well.)
3. Low-level solutions
   - 3.1. Pure software
   - 3.2. Special hardware used by software
   - 3.3. Problem in either case: lack of clarity of code when using; potential for error due to failure to use/misuse because of programmer error or laziness
   - 3.4. Not directly usable for distributed systems
4. Semaphores - primitive proposed by Dijkstra
   - 4.1. Actually implemented on top of lower level solution
   - 4.2. Types
     - 4.2.1. Binary - initial value = 1
       Definitions: P(s) ::= atomically do the following: while (s <= 0) ; s --;
       V(s) ::= atomically do the following: s ++
     - 4.2.2. Counting - initial value = number of P() operations allowed before a wait is required.
       Definitions: Sane as bubart
     - 4.2.3. Both of the above rely on busy-waiting. A third type allows a process to block on the semaphore and keeps a list of waiting processes. In this case, if the value of the semaphore is negative its absolute value is the number of blocked processes.
       Definitions: P(s) ::= atomically do the following: s --; if (s < 0) block
       V(s) ::= atomically do the following: s ++; if (s <= 0) unblock one waiter
     - 4.2.4. The queue in 4.2.3 is defined as non-FIFO.
   - 4.3. Much clearer, but again, failure to use/misuse easily possible due to programmer error or laziness
   - 4.4. Not directly usable for distributed systems
5. Message-Passing
   - 5.1. Key idea: shared variable “owned” by a single process; other processes access via messages to that process; owner ensures mutual exclusion
   - 5.2. Not as prone to misuse since shared variable cannot be accessed directly
   - 5.3. Usable in distributed systems - but requires a fair amount of overhead if used in a shared memory system (e.g. for threads)
6. Abstract Monitors
   - 6.1. Abstraction implemented on top of lower-level solution. Not implemented in “pure” form in any major programming language.
   - 6.2. A monitor must be explicitly declared as such.

6.3. When this is done, guarantees mutual exclusion for all entries. When a process is executing a monitor entry, no other process can execute any entry on the same object. (If two objects use same monitor "class", each is independent of the other in terms of mutual exclusion)

6.4. A monitor can declare any number of conditions. A process using the monitor can wait on or signal any condition. A signal on a condition that has no waiters is ignored.

6.5. FIFO behavior guaranteed for initial entry and for condition waits if multiple processes are waiting on the same condition. A condition wait that has been signaled has priority over an entry from the outside.

7. Java Monitors

7.1. Based on abstract monitors.

7.2. Every object has its own monitor. (Capability inherited from common base class `java.lang.Object`)

7.3. To avoid high-overhead of locking, explicit use of `synchronized` required to get mutual exclusion.

7.4. A monitor has a single implicit condition. A process using the monitor (i.e. inside a block declared `synchronized`) can `wait()` on or `notify()` the monitor (which is an implicit wait/signal on the condition). If there are no waiters, `notify()` is ignored.

7.5. FIFO behavior is not guaranteed for `synchronized` or if there are multiple waiters when a `notify()` is done. If `notify()` is done, there is no guarantee that the awakened waiter will be the next to enter the monitor if others are waiting to enter.

7.6. Two other "Java-isms"

7.6.1. `InterruptedException`

7.6.2. `notifyall()`.