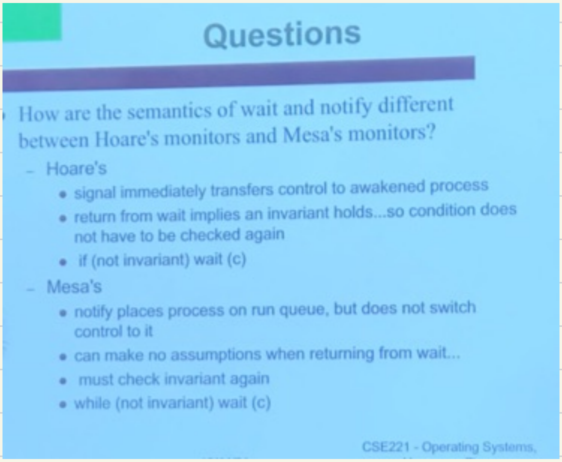
10/14

- C. A. R. Hoare, Monitors: An Operating System Structuring Concept, Communications of the ACM, Vol. 17, No. 10, October, 1974, pp. 549-557.

Q: What are "monitor invariant" I and "condition" B, and why are they important in the discussion of monitors?

- B. W. Lampson and D. D. Redell, Experience with Processes and Monitors in Mesa, Communications of the ACM, Vol. 23, No. 2, February 1980, pp. 105-117.

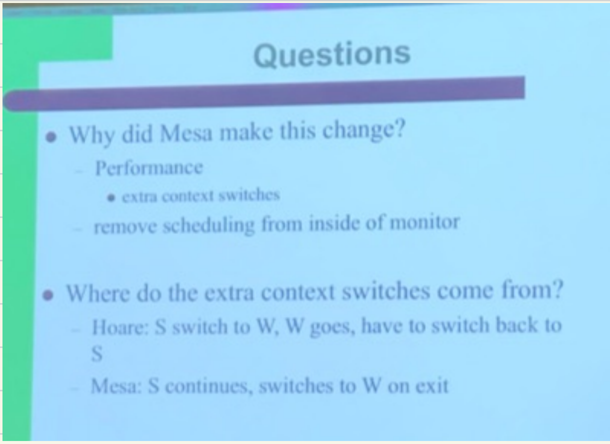
Q: Compare and contrast synchronization in Java with Hoare monitors and Mesa monitors.



Questions

- How are the semantics of wait and notify different between Hoare's monitors and Mesa's monitors?
  - Hoare's
    - signal immediately transfers control to awakened process
    - return from wait implies an invariant holds...so condition does not have to be checked again
    - if (not invariant) wait (c)
  - Mesa's
    - notify places process on run queue, but does not switch control to it
    - can make no assumptions when returning from wait...
    - must check invariant again
    - while (not invariant) wait (c)

CSE221 - Operating Systems,



Questions

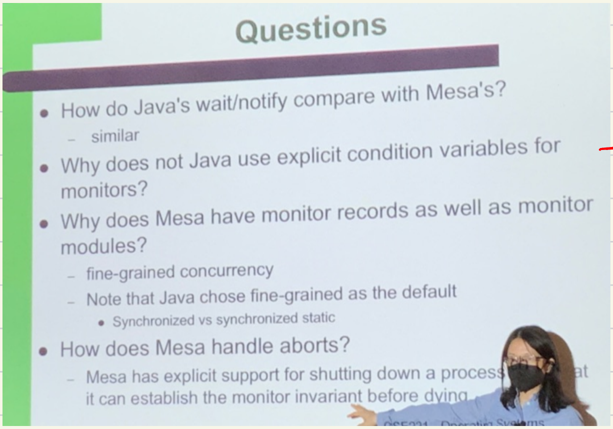
- Why did Mesa make this change?
  - Performance
    - extra context switches
  - remove scheduling from inside of monitor
- Where do the extra context switches come from?
  - Hoare: S switch to W, W goes, have to switch back to S
  - Mesa: S continues, switches to W on exit

Java vs Mesa [PICs, on Wikipedia]

Mesa has multiple Queue.

Java has only one implicit one.

No Explicit.

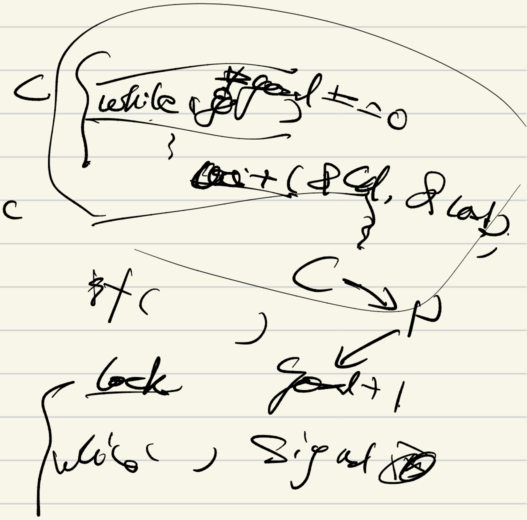


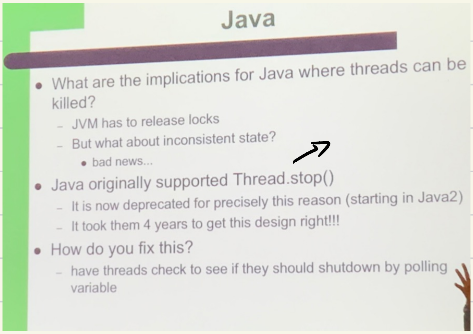
Questions

- How do Java's wait/notify compare with Mesa's?
  - similar
- Why does not Java use explicit condition variables for monitors?
- Why does Mesa have monitor records as well as monitor modules?
  - fine-grained concurrency
  - Note that Java chose fine-grained as the default
    - Synchronized vs synchronized static
- How does Mesa handle aborts?
  - Mesa has explicit support for shutting down a process [illegible] at it can establish the monitor invariant before dying

CSE221 - Operating Systems

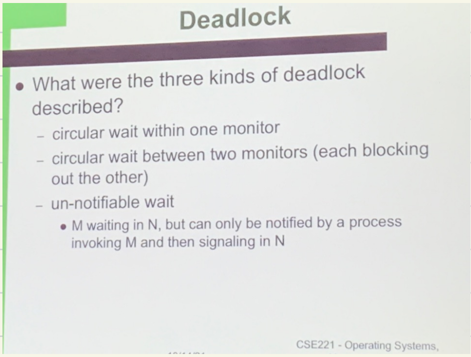
→ High level
free programmer from
this trivel control





Java

- What are the implications for Java where threads can be killed?
  - JVM has to release locks
  - But what about inconsistent state?
    - bad news...
- Java originally supported Thread.stop()
  - It is now deprecated for precisely this reason (starting in Java2)
  - It took them 4 years to get this design right!!!
- How do you fix this?
  - have threads check to see if they should shutdown by polling variable

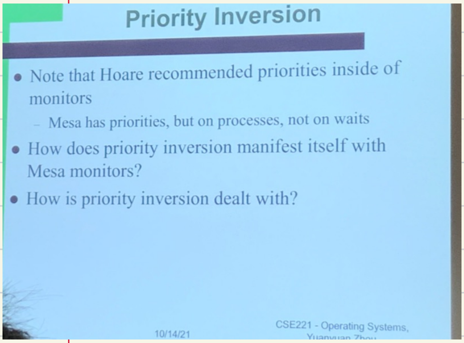


Deadlock

- What were the three kinds of deadlock described?
  - circular wait within one monitor
  - circular wait between two monitors (each blocking out the other)
  - un-notifiable wait
    - M waiting in N, but can only be notified by a process invoking M and then signaling in N

CSE221 - Operating Systems,

Priority Inversion



Priority Inversion

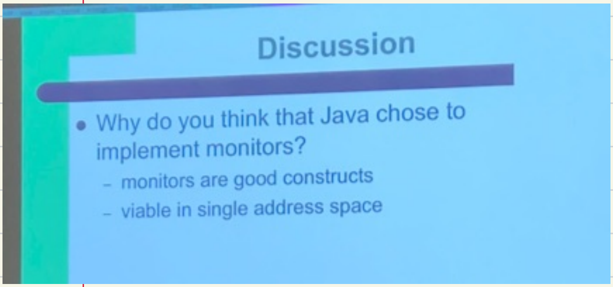
- Note that Hoare recommended priorities inside of monitors
  - Mesa has priorities, but on processes, not on waits
- How does priority inversion manifest itself with Mesa monitors?
- How is priority inversion dealt with?

10/14/21

CSE221 - Operating Systems, Yuanyuan Zhou

① High priority process has to wait for lower process to release the resource

② However, it can't, because it doesn't have much resource to run



Discussion

- Why do you think that Java chose to implement monitors?
  - monitors are good constructs
  - viable in single address space