**Chapter 7 Deadlocks**

Critical Section

* A section of code in which process access and modifies **shared variables**

Mutual Exclusion

* **mutual exclusion** is a program object that prevents simultaneous access to a shared resource.

Why process need to communicate?

* To synchronize their executions
* To exchange data and information

Rules to from Critical Sections

* No two process may be simultaneously inside their CS (mutual exclusion)
* No assumption is made about relative process speeds or number of CPUs
* A process outside a CS should not block other processes
* No process should wait forever before entering its CS

Starvation

* Definition: Identify delaying the scheduling of a process in favor of other processes
* Cause: Usually a bias in a system scheduling policies (a bad scheduling algorithm)
* Solution: Implement some form of aging

Deadlocks

* Two or more process are blocked waiting for an event that will never occur
* Generally, A waits for B to do something and B is waiting for A
* Both are not doing anything so both events never occur

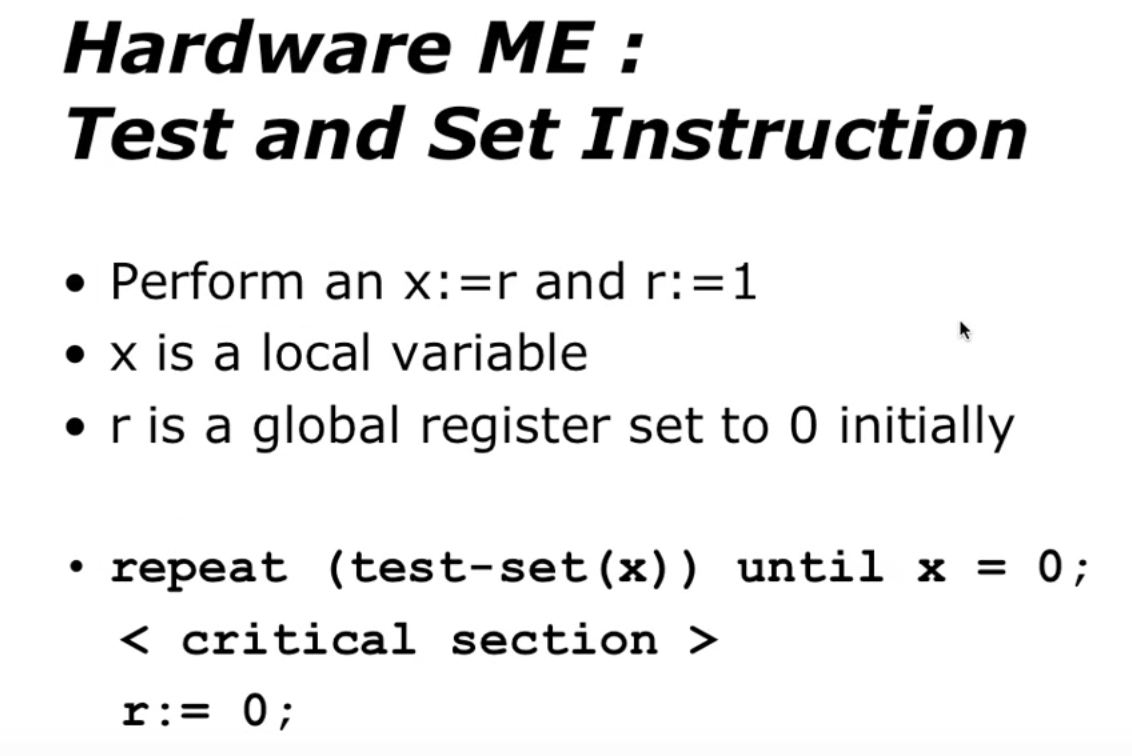
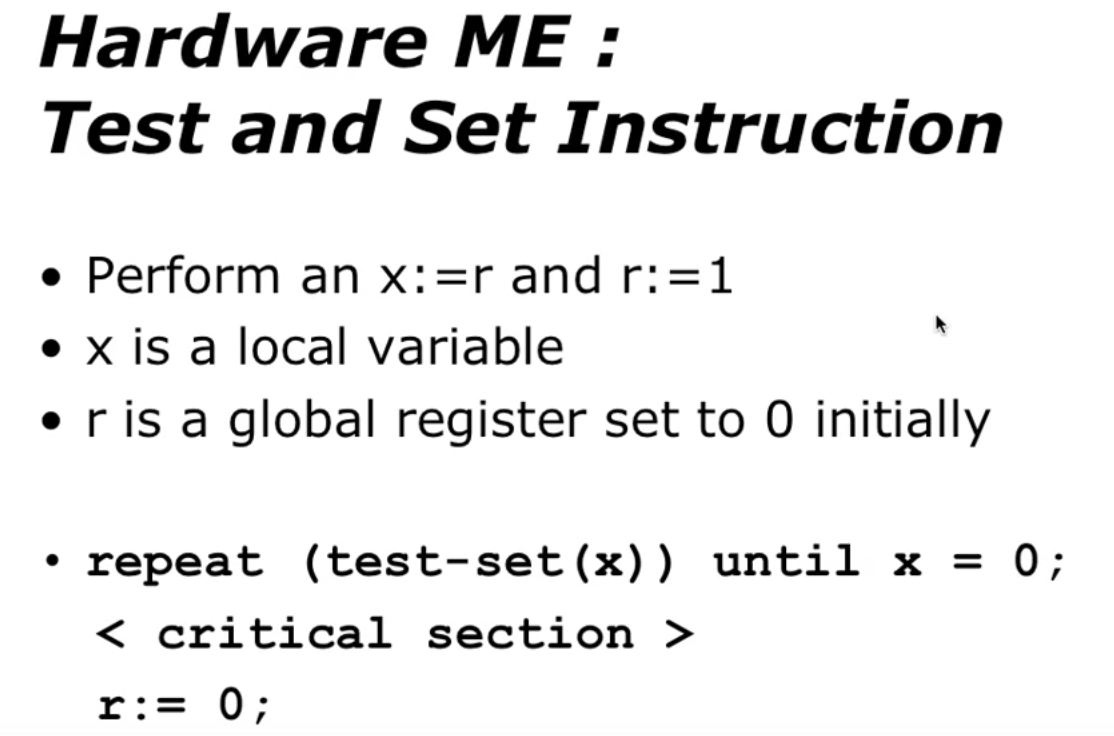
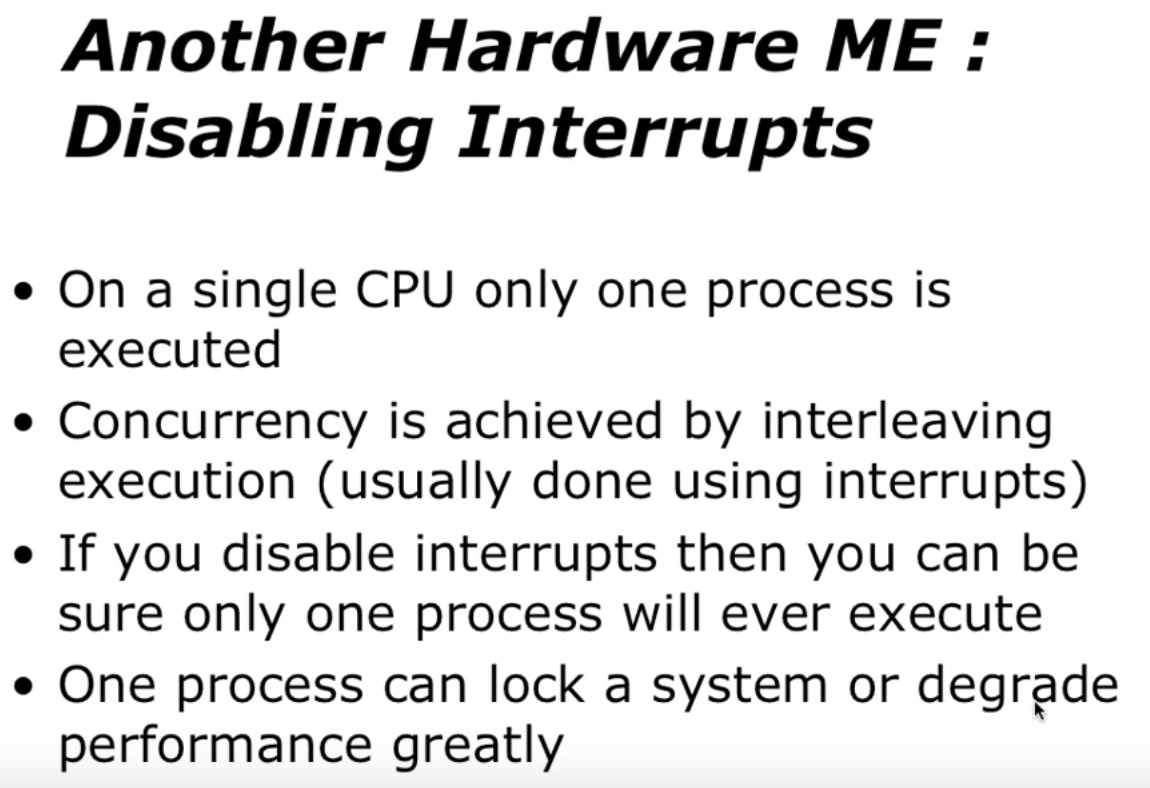
How to Implement Mutual Exclusion

* Three possibilities
* Application: programmer builds some method into the program
* Hardware: special h/w instructions provided to implement ME
* OS: provides some services that can be used by the programmer
* All schemes rely on some code for
* enter\_critical\_section, exit\_critical\_section

Application Mutual Exclusion

* Application Mutual Exclusion is
* Implemented by the programmer
* Hard to get correct, and very inefficient
* All rely on some form of busy waiting (process tests a condition, set a flag, and loops while the condition remains the same)

Hardware Mutual Exclusion Characteristics

* Advantage
* Can be used by a single or multiple process (with shared memory
* Simple and therefore easy to verify
* Can support multiple critical sections
* Disadvantages
* Busy waiting is used
* Starvation is possible
* Deadlock is possible

Mutual Exclusion Through OS

* Semaphores
* Message passing

What is Deadlock?

* Process Deadlock
* A process is deadlock when it is waiting on an event which will never happen
* System Deadlock
* A system is deadlock when one or more processes are deadlock

Necessary conditions for a Deadlock

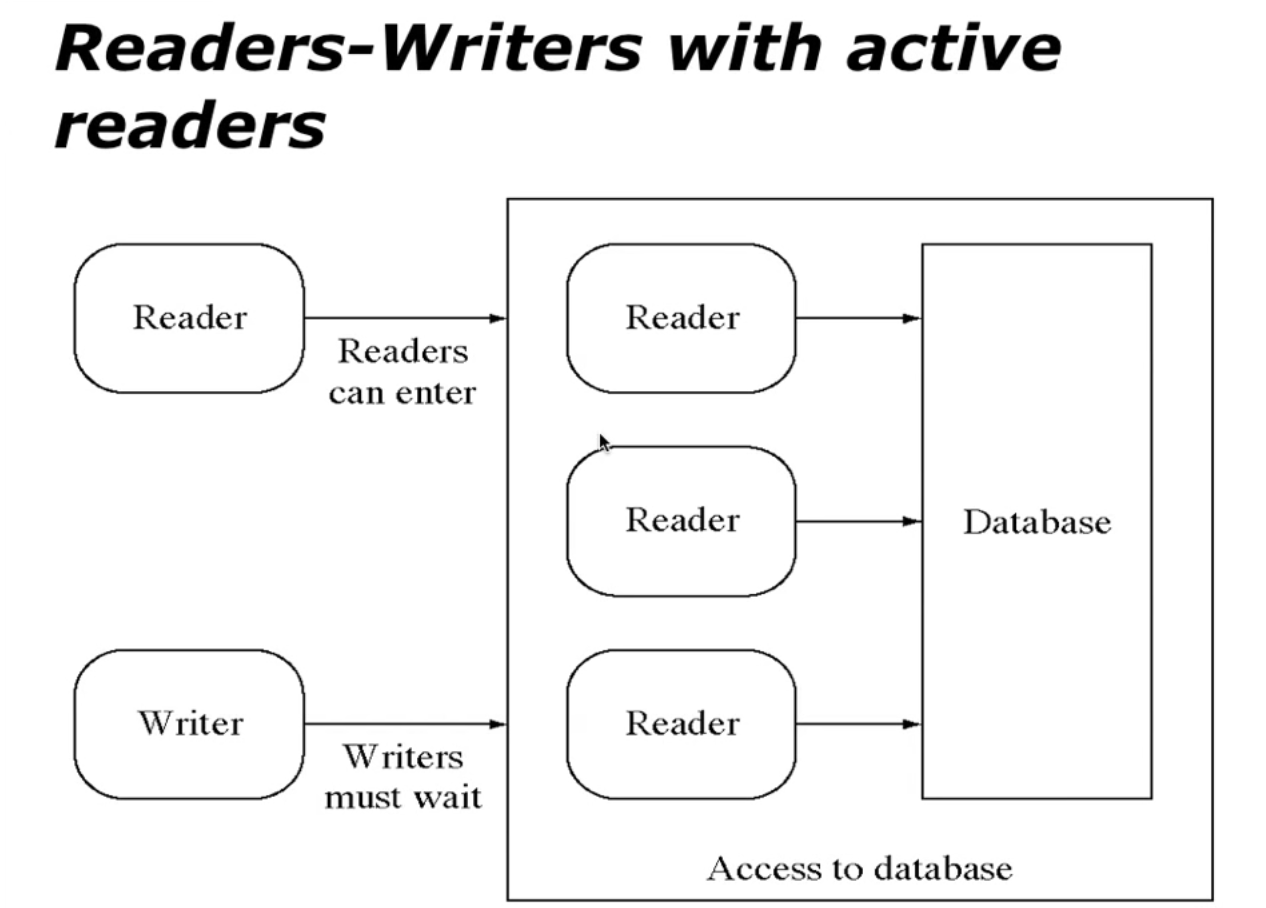
* Mutual Exclusion
* Shared resources are used in a mutually exclusive manner
* Hold & Wait
* Process hold onto resources they already have while waiting for the allocation of other resources
* No preemption
* Resources can not be preempted until the process releases them
* Circular Wait
* A circular chain of processes waits in which each process holds resources wanted by next process in the chain

No Dead Situation

* If you can prevent at least one of the necessary, deadlock conditions then you won’t have a DEADLOCK

Ways of Handling Deadlock

* Deadlock Prevention
* Remove the possibility of deadlock occurring by denying one of the four necessary conditions:



* Mutual Exclusion
* Hold & Wait
* No preemption
* Circular Wait
* Deadlock Detection
* Deadlock Avoidance
* Deadlock Recovery

Readers – Writers Problem

* Any number of reader activities and writer activities are running
* At any time, a reader activity may wish to read data
* At any time, a writer activity may want to modify the data
* Any number of readers may access the data simultaneously
* During the time a writer is writing, no other reader or writer may access the shared data

1. The first reader – writer problem, requires that no reader will kept waiting unless a writer has obtained access to shared data
2. The second reader – writer problem, requires that once a writer is ready, no new readers may start reading
3. In a solution to the first case, writers may starve. In a solution to the second case readers may starve

First Reader – Writer Solution

* **Readcount** counter keeps track of how many processes are currently reading
* **Mutex** semaphore provides mutual exclusion for updating readcount
* **wrt** semaphore provides mutual exclusion for the writers; it is also used by the first or last reader that enters or exist the CS