Chapter 6 2/26/24

cooperating process - process that can affect or be affected by other processes executing in a system, can share code or data through logical address space, or through shared memory or message passing

6.1 concurrent - dealing with many process at once, Parallism - doing multiple processes at once

race condition - several processes access & manipulate same data concurrently & the outcome depends on order in which access happened

To stop race condition we must make sure that only one process can manipulate variable count.

6.2 The critical Section Problem

critical section - one process may be accessing & updating data shared with one other processes. When one process is executing in critical section No other process can be execute in its critical section

Critical Section Problem - design a protocol that process can use to synchronize their activity to share data cooperatively

entry section - the code that request permission to enter critical section

Solution to Critical section problem

1. Mutal Exclusion - Process P can be executing in critical section then No other Process can execute there
2. Progress - No process is executing in critical section & some process wants to enter Critical sections, then only processes not executing in remainder section can join in and decide, which process will enter critical section Next. & cannot be postponed for ever

Remainder Section - rest of code where process can execute without accessing trouble

```
While (true) {
    entry section
        critical section
    exit section
        remainder section
```

3. Bounded Waiting - bound or limit on the number of times that other processes are allowed to enter their critical section after a process has made a request to enter its critical section & before request is granted

Chap 6 Continued

6.3 Peterson's Solution Algo in book

6.4 Hardware Support for Synchronization

Memory model - how a computer architecture determines what memory guarantees it will provide to an application, 2 Categories

Storngly ordered - memory modification on one processor, visible to all other processors, immediate

weekly ordered - memory modifications on one processor may not be visible to other processors right away

memory barrier or memory fences - instructions that force changes in memory to be to be visible to other threads so memory modifications are visible to thread on other processors, So if instructions were reordered, memory barrier ensures all loads & reads are done, before any load & reads are done in future, & visible to other processes before future load or store operations are preformed

atomic variables - provides atomic operations on integers & booleans, operations completed on atomic variables without interuption

6.5 Mutex Locks - mutual exclusion, used to prevent critical section & avoid race conditions, process must aquire the lock before entering a critical section, releases lock when exiting critical section

spin Lock - process spins the block while waiting for lock to be avilble, or called busy waiting - loop continously waiting for lock to be avilble

6.6 Semaphores - S integer variable only accessed through standard atomic operations wait() & signal(), semaphore is set to number of resources avilable, each process wanting to use resource calls wait() which decrements count, releases resource calls signal() increments count. Semphore = 0 all resources being used, process will block untill the cant greter then 0 saying resources avilble

↑ wait state until condition met

Solve busy waiting in Semaphores - When calling wait semaphore not positive, instead of bussy waiting the process can suspend itself

6.7 Monitors

monitor type - Abstract data type - programer defined operations that provided with mutual exclusion within the monitor

Chap 6 Continued

6.8 Liveness - set of properties that a system must satisfy to ensure process make progress during execution. Process waiting forever is a "liveness failure"

Deadlock - 2 or more processes cant proceed because each process is waiting for a resource held by the other process to be released or waiting for other process to take an action, but the other proces is waiting for og process to take an action

Deadlock better Def - every process in set is waiting for an event that can be only caused by other process in set

priority inversion - low priority task hlds a resource Needed by high priority process. Example 3 processes L, M, H Priority order L < M < H H need semaphores, used by L. H waits For L to do done but bitch ess M is runnable ∴ prevents L ∴ steals H semaphores

priority-inheritance protocol - Fixes priority inversion problem processes using a resources needed by higher priority process inherit high priority from high priority proces so they dont preempted by bitch ess M. When they finish back to Normal priority

Uncontended - CAS Compare & Swap Faster than traditional synchronization

Moderate contention - CAS will possibly be faster than traditional synchronized

High contention - traditional will be ultimately faster than CAS approach