Advanced Synchronization

Bringing It All Together

- Processes
  - Abstraction for protection
  - Define address space
- Threads
  - Share (and communicate) through global and static data, share the heap, each has its own stack and full use of the registers
  - Race conditions may be a problem!
- CPU Schedulers
  - May pre-empt a process or thread at any time
- Ensuring correctness (OR eliminating race conditions and deadlock)
  - Safety and liveness
  - Atomic instructions
  - Synchronization: mutual exclusion, counted resources…
  - Locks, semaphores, monitors, transactions, conservative two-phase locking
  - Common patterns: Bounded Buffer, Dining Philosophers, Readers/Writers

Vocabulary

deadlock
rollback
One Big Lock
power cycle
write-behind
durability

serializability
commit
transactions
write-ahead
memory
transaction log

disk
idempotent
fine-grained locking
coarse-grained locking
multi-object synchronization
conservative two-phase locking

Readers and Writers

A Different Type of Problem

- We've looked at problems where we protect shared data by only allowing one thread in the critical section at a time
- Is this always appropriate? When might we want to let more threads access shared data at once?

Readers/Writers Problem

- Data is shared among several threads
  - Some only read
  - Some only write
- To get correct results, we allow *multiple readers* at a time, but only *one writer* at a time
- How can we control access to the object to permit this protocol?

Correctness Criteria

- Each read or write of the shared data *must* happen within a critical section
- Guarantee mutual exclusion for writers
- Allow multiple readers to execute in the critical section at once
- Allow one writer (and no readers) to execute in the critical section at once

Readers and Writers: Monitor Solution

- What methods do we need?
- How many locks?
- How many condition variables?
- What should we name them?
- Any other variables?

- Assume we're going to say <read> and <write> for accesses to the shared data.

Readers and Writers: Monitor Solution

Variables:

read(){

}

write(){

}

Readers and Writers: Monitor Solution

Variables:

read(){

}

write(){

}

Is our solution fair?

A. Yes
B. No, favors readers
C. No, favors writers

Understanding Our Solution

It works, but it favors readers over writers

- Any reader blocks all writers
- All readers must finish before a writer can start
- Last reader will wake any writer, but a writer wakes all readers and writers
- If a writer exits and a reader goes next, then all readers that are waiting will get through

Alternative Semantics

- It may be that you would like a writer to enter its critical section as soon as possible.
- How could we implement that?