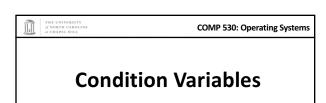
COMP 530: Operating Systems



Don Porter

Portions courtesy Emmett Witchel

No, but what is the "right" way to build a parallel program?

 People are still trying to figure that out.

 Compromises:

 between making it easy to modify shared variables AND
 restricting when you can modify shared variables.
 between really flexible primitives AND
 simple primitives that are easy to reason about.

Synchronization
• Now that you have seen locks, is that all there is?

COMP 530: Operating Systems

 Moving Beyond Locks
 Synchronizing on a condition.

 When you start working on a synchronization problem, first define the mutual exclusion constraints, then ask "when does a thread wait", and create a separate synchronization variable representing each constraint.

 Bounded Buffer problem – producer puts things in a fixed sized buffer, consumer takes them out.

 What are the constraints for bounded buffer?

 1) only one thread can manipulate buffer queue at a time (mutual exclusion)

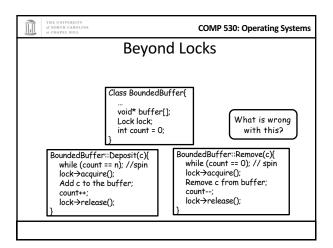
 2) consumer must wait for producer to fill buffers if none full

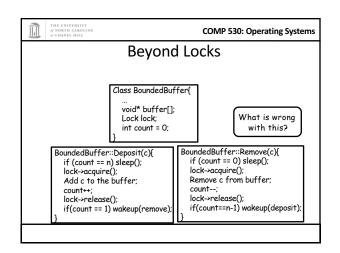
- 3) producer must wait for consumer to empty buffers if all full

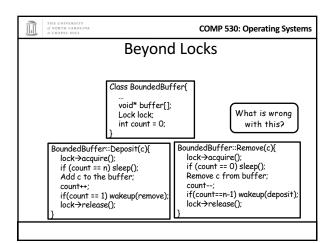
(scheduling constraint)

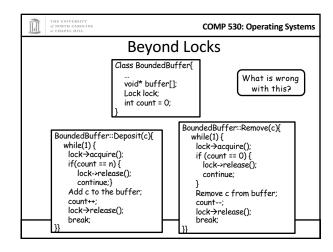
(scheduling constraint)

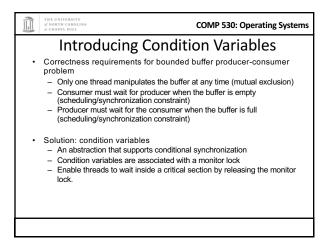
COMP 530: Operating Systems Beyond Locking · Locks ensure mutual exclusion Bounded Buffer problem - producer puts things in a fixed sized buffer, consumer takes them out. - Synchronizing on a condition. Class BoundedBuffer{ void* buffer[]; What is wrong Lock lock; with this? int count = 0; BoundedBuffer::Remove(c){ BoundedBuffer::Deposit(c){ lock→acquire(); while (count == 0); // spin lock→acquire(); while (count == n); //spin Add c to the buffer; Remove c from buffer; count++; lock→release(); count-lock→release();

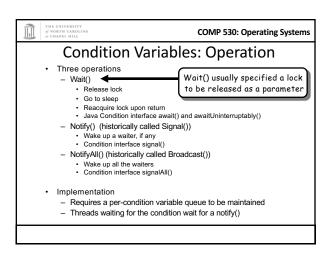


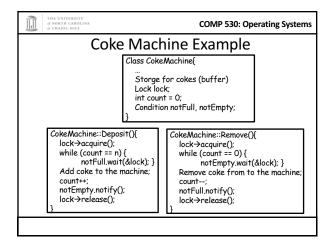


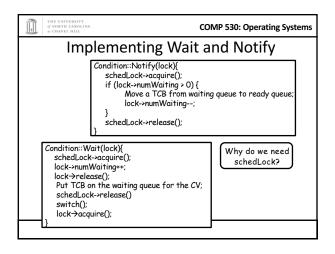














COMP 530: Operating Systems

Using Condition Variables: An Example

- Coke machine as a shared buffer
- · Two types of users

 - Producer: Restocks the coke machine
 Consumer: Removes coke from the machine
- · Requirements
 - Only a single person can access the machine at any time
 - If the machine is out of coke, wait until coke is restocked
 - If machine is full, wait for consumers to drink coke prior to restocking
- · How will we implement this?
 - What is the class definition?
 - How many lock and condition variables do we need?



COMP 530: Operating Systems

Word to the Wise...

- Always wait and notify condition variables with the mutex held.
- · Period.
 - Fine print: There are cases where notification outside of a lock can be safe, but the code tends to be fragile, errorprone, and easy for another developer to break.
 - In many cases you can lose notifications and hang (liveness)
 - Moreover there is no clear advantage to breaking this convention. So just don't do it.



COMP 530: Operating Systems

Summary

- Non-deterministic order of thread execution → concurrency problems
 - Multiprocessing
 - A system may contain multiple processors → cooperating threads/processes can execute simultaneously
 - Multi-programming
 - · Thread/process execution can be interleaved because of time-slicing
- · Goal: Ensure that your concurrent program works under ALL possible
- Define synchronization constructs and programming style for developing concurrent programs
 - Locks → provide mutual exclusion
 - Condition variables \rightarrow provide conditional synchronization