University of BRISTOL

Computer Systems B

COMS20012

Introduction to Operating Systems and Security

bristol.ac.uk

University of BRISTOL

Synchronization primitives

bristol.ac.uk

University of BRISTOL

Look back at Computer Systems A

- This is a quick refresher

bristol.ac.uk

Locks

- Obtain a resource for exclusive use
  - Acquire/lock: get the resource
  - Release/unlock: give up the resource
- Use case
  - Need to arbitrate exclusive access to a resource
  - If resource is unavailable want to wait for the resource
  - **Same agent acquire/release access to the resource**

Spinlock

- Very simple locking mechanism
- **Busy wait** for resource to be available
  - Atomically test if a resource is available and get it
  - If it's not available try again.
- Requirements
  - Require some hardware support (disabling interrupts or some atomic instructions such as TAS or CAS)
    - TAS: test-and-set
    - CAS: compare-and-swap
- Should be used only for a short duration
  - Busy wait waste CPU resources

Semaphore

- Counting and locking mechanism (shared counter)
  - Value always >= 0
  - P decrease the value (proberen which means “to test” in Dutch)
  - V increase the value (verhogen, which means “increase” in Dutch)

Semantic

- V increase the counter
- P wait for the counter to go positive and decrement

Semaphore usage

- Binary semaphore (similar to a lock)
  - Initialize the semaphore to 1
  - Semaphore will only take value 0/1
  - P acquires resource
  - V releases resource
- Counting semaphore
  - N interchangeable resources are available
  - Initialize the semaphore to N
  - P uses one of N resource
  - V frees resource
  - Allow up to N simultaneous users

Condition variable

- Allow to run only when a condition about the world is true
- Always paired with a lock
- API
  - **cv_create/cv_destroy** (create/destroy a condition variable)
  - **cv_wait** block and sleep until the condition becomes true
  - **cv_broadcast** wake all thread waiting on this condition
  - **cv_signal** wake a single thread waiting on this condition
- Use case
  - Want to run when a condition is true
  - Not necessarily exclusive access
  - Condition is typically simple

Condition variable semantic

- Usage
  - Acquire lock
  - Check condition
  - If you need to wait on condition cv_wait
  - Once condition is true decide if you want to cv_broadcast or cv_signal
  - Release lock
- Semantics
  - Hoare semantic: if you wait on a condition, when you wake up you are **guaranteed the condition is true**.
  - Mesa semantic: **No guarantees when you wake**; someone else may have beaten you.
  - **OS/161 uses Mesa** semantic be careful.

Kernel Synchronization Similarities

- Can use all the same primitives
- Same principles: critical sections, deadlock etc.
- Deadlock are easier to debug than race conditions
- Same requirements:
  - Only one thread in a critical section
  - Must make forward progress
  - Activity outside a critical section cannot block the critical section.
    - e.g. careful when using nested locks
  - Critical sections are short.
- Desirable properties
  - Fair: if several threads are waiting let them in eventually
  - Efficient: don't use significant resources when waiting (e.g. busy wait)

Kernel Synchronization Differences

- Somewhat similar to synchronizing with a server
  - Again, go back to Computer Systems A
- Differences from synchronizing with normal user code
  - Must synchronize with hardware
  - Performing operation on behalf of “someone else” (e.g. you don’t necessarily know all the resources a user thread will want)
  - Must avoid deadlock at all cost (**finding deadlock and killing processes is not ok!**)

University of BRISTOL

Thank you

bristol.ac.uk