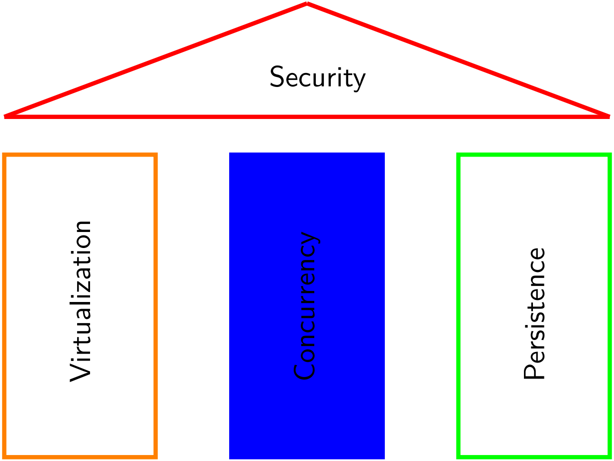
CS323 Operating Systems

Concurrency Summary

Mathias Payer and Sanidhya Kashyap

EPFL, Fall 2021

Concurrency

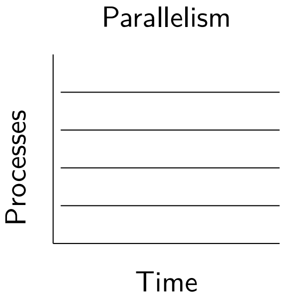


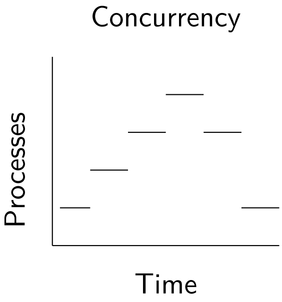
Concurrency topics

- Abstraction: locks to protect shared data structures
- Mechanism: interrupt-based locks
- Mechanism: atomic hardware locks
- Busy waiting (spin locks) versus wait queues
- Condition variables
- Semaphores
- Signaling through condition variables and semaphores

Difference parallelism and concurrency

- ***Parallelism:*** multiple threads (or processes) working on a single task using multiple CPU cores (i.e., stuff happens at the same physical time)
- ***Concurrency:*** tasks can start, run, and complete in overlapping time periods (i.e., tasks run at the same virtual time)





Locks: basic idea

- Requirements: mutual exclusion, fairness, and performance
  - ***Mutual exclusion***: only one thread in critical section
  - ***Fairness***: all threads should eventually get the lock
  - ***Performance***: low overhead for acquiring/releasing lock
- Lock implementation requires hardware support
  - ... and OS support for performance

Lock types

- Interrupts
- (Buggy) software lock
- (Buggy) Peterson's lock
- Spin lock: test-and-set
- Spin lock: compare-and-swap
- Queue lock

Lock best practices

- When acquiring a lock, recheck assumptions
- Ensure that all shared information is refreshed (and not stale)
- Multiple threads may wake up and race for the lock (i.e., loop if unsuccessful)

Lock summary

- Locks enforce mutual exclusion for critical section (i.e., an object that can only be owned by a single thread)
- Trade-offs between spinlock and queue lock
  - Time lock is held
  - Contention for lock
  - How many concurrent cores execute
- Locking requires kernel support or atomic instructions
  - **test-and-set** atomically modifies the contents of a memory location, returning its old value
  - **compare-and-swap** atomically compares the contents of a memory location to a given value and, iff they are equal, modifies the contents of that memory location to a given new value.

Condition Variables (CVs)

- A CV allows a thread to wait for a condition
  - Usually implemented as queues
  - Another thread signals the waiting thread
- API: `wait`, `signal` or `broadcast`
  - `wait`: wait until a condition is satisfied
  - `signal`: wake up one waiting thread
  - `broadcast`: wake up all waiting threads

Locks vs. CVs

- Lock an object that can only be owned by a single thread
  - Enforces mutual exclusion
  - `acquire(lock_t *lck)`: acquire the lock, wait if needed
  - `release(lock_t *lck)`: release the lock
- CVs allow a thread to wait for an event (condition)
  - Lock for mutual exclusion, condition to signal event has passed
  - `wait(cond_t *cond, lock_t *lck)`: wait until cond is true
  - `signal(cond_t *cond, lock_t *lck)`: signal one thread
  - `broadcast(cond_t *cond, lock_t *lck)`: signal all threads

Semaphores

- A semaphore extends a CV with an integer as internal state
- `int sem_init(sem_t *sem, unsigned int value)`: creates a new semaphore with `value` slots
- `int sem_wait(sem_t *sem)`: waits until the semaphore has at least one slot
- `int sem_post(sem_t *sem)`: increments the semaphore (and wakes one waiting thread)
- `int sem_destroy(sem_t *sem)`: destroys the semaphore and releases any waiting threads

Semaphores/spin locks/CVs are equivalent

- Each can be implemented through a combination of the others
- Depending on the use-case, performance will vary
  - How often is the critical section executed?
  - How many threads compete for a critical section?
  - How long is the lock taken?

Book chapters

- Concurrency/Locking: OSTEP 28-30
- Concurrency/Semaphores: OSTEP 30-32

Concurrency summary

- Spin lock, CV, and semaphore synchronize multiple threads
  - Spin lock: atomic access, no ordering, spinning
  - Condition variable: atomic access, queue, OS primitive
  - Semaphore: shared access to critical section with (int) state
- All three primitives are equally powerful
  - Each primitive can be used to implement both other primitives
  - Performance may differ!
- Synchronization is challenging and may introduce different types of bugs such as atomicity violation, order violation, or deadlocks.