Problem Set 1

Q3 :

```
gettimeofday();      -> actually a system call.
loop{ procedure call() }   -> needs to run many iterations to
                              make the time syscall itself negligible.
gettime of day();
```

10/16/23

Locks

→ synchronization primitive that provides mutual exclusion

→ APIs: lock_acquire(); // access to shared data (critical section) lock_release();

→ Properties of a lock

common types of property

① Safety: nothing bad ever happens → no 2 threads can hold the lock at the same time

② Liveness: something good eventually happens → progress: a thread can grab the lock if it's free

③ Fairness: be fair! → bounded waiting: there's an upperbound to your wait time. (can't keep skipping over a thread)

Types of Locks.

```
lock_acquire();
global_x++;
lock_release();
```

① Spinlock

lock waiter either in ready or running state

→ while loop checks lock status until it's free

→ implemented w/ atomic instr.

→ long critical section

⇒ long wait time

→ short critical section

$\not\Rightarrow$ short wait time?

not always. → I/O.

→ many waiters (lock contention)

② Sleeplock

blocking state

→ blocks/sleeps until the lock is free.

impl. w/ a list of waiters & sleep/notify mechanism.

→ ↳ requires context switch (may cause large overhead)

short wait time / short critical section

- still needs to pay the context switch cost

How about long wait time?

Example usage:

xk fs ops are protected w/ sleeplock (inode)

★ Some cases where you have to use spinlock

→ interrupt handlers

Monitors

→ design pattern & synchronization primitive that coordinate threads based on events.

- sleep/wake up abstraction

→ consisted of

- conditions : events threads are synchronizing on → accessed & updated by multiple threads
- locks : protect access to the conditions & condition variables
- (list of waiters) condition variables : track waiters of a condition, implement sleep & wakeup

→ cv_wait : put the calling thread to waiter list, blocks the thread & releases the lock; when unblocked, acquires lock & then returns

→ cv_signal : wake up a waiter, remove from waiter list (Blocking → ready)

→ cv_broadcast : wake up all waiters. used when the condition is changed & might unblock multiple waiters.