Concurrency #3:

→ Ticket lock
→ Queued lock (w/ park/unpark)
→ Intro to CVs

① Review Ticket Lock

point: fairness (no starvation)

(overflow problem?)

Eval of locks:

performance:
- n threads, 1 CPU
- n thr, n CPUs

fairness:
- guaranteed to acquire lock?

② How to avoid spinning?

(a) yield : release CPU, move to end of run queue

(b) park(), unpark(thread)

---

③ Condition Variables

Def: ⇒ cond_t ⇒ queue ⇒ paired w/ a lock

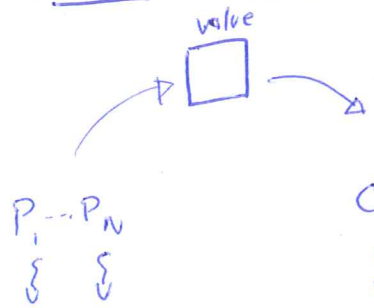
operations
- → wait() ⇒ put self to sleep on queue, release lock
- → signal() ⇒ wake exactly 1 waiter (if w exists) (assumes lock is held)

usually paired with another variable (e.g. int) to track state

④ Example: The join problem

code snippet in Lecture-Condvar/ (main-join-

⑤ intro: producer/consumer problem



for next time

Handouts / threads-hw-primitives-2.pdf

```
typedef struct __lock_t {
    int         flag;   // state of lock: 1=held, 0=free
    int         guard;  // use to protect flag, queue
    queue_t     *q;     // explicit queue of waiters
} lock_t;

void lock_init(lock_t *lock) {
    lock->flag = lock->guard = 0;
    lock->q    = queue_init();
}

void lock(lock_t *lock) {
    while (xchg(&lock->guard, 1) == 1)
        ; // spin
    if (lock->flag == 0) { // lock is free: grab it!
        lock->flag = 1;
        lock->guard = 0;
    } else {               // lock not free: sleep
        queue_push(lock->q, gettid());
        lock->guard = 0;
        park();            // put self to sleep
    }
}

void unlock(lock_t *lock) {
    while (xchg(&lock->guard, 1) == 1)
        ; // spin
    if (queue_empty(lock->q))
        lock->flag = 0;
    else
        unpark(queue_pop(lock->q));
    lock->guard = 0;
}
```

1) spins on guard : why? (what is protected?)

2) still spins: why better than simple spin lock?

3) unlock: on unpark(), no ~~ref~~ setting of flag ⇒ 0 ⇒ why?

4) race condition

```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include "mythreads.h"

pthread_cond_t  c = PTHREAD_COND_INITIALIZER;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
int done = 0;

void *
child(void *arg) {
    printf("child\n");
    Mutex_lock(&m);
    done = 1;
    Cond_signal(&c);
    Mutex_unlock(&m);
    return NULL;
}
int main(int argc, char *argv[]) {
    pthread_t p;
    printf("parent: begin\n");
    Pthread_create(&p, NULL, child, NULL);
    Mutex_lock(&m);
    while (done == 0)
        Cond_wait(&c, &m); // releases lock when going to sleep
    Mutex_unlock(&m);
    printf("parent: end\n");
    return 0;
}
```