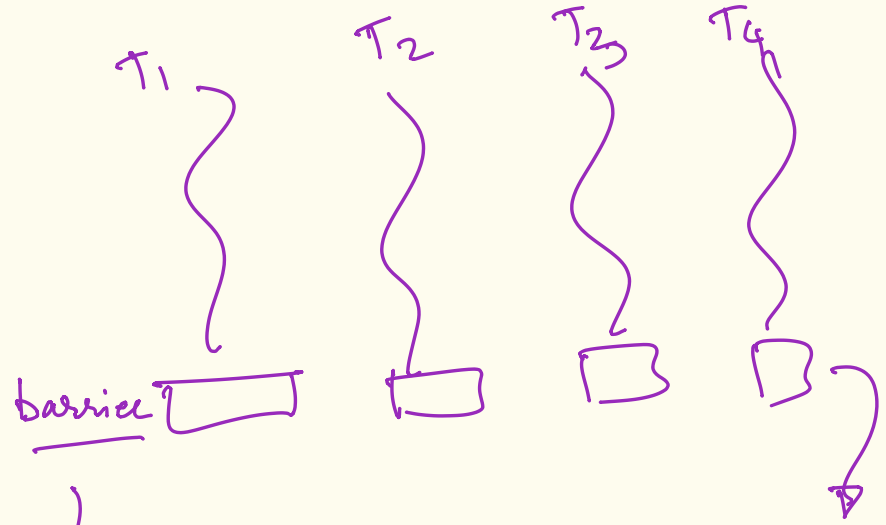
CS744 Lecture 16

midsem

~~Oct.~~ 11th Oct.

* synchronization.
  - spinlocks, mutexes, ...
    - spinlocks — busy locks
    - mutexes — & sleep locks

* condition variables — generalized mutexes
  - conditions to wait / wakeup
  - semaphores — counting-based sync. primitives
    - up — V
    - down — P
      - atomically update a shared variable
      - if shared == 0 V sleep
  - barriers —
  - reader/writer locks —



wait till all threads reach this point of execution.

once all threads reach barrier, barrier is open.

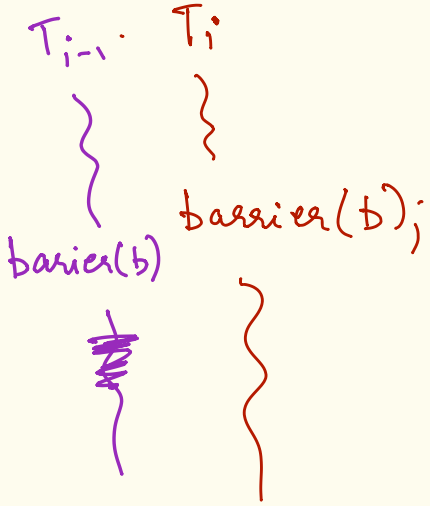
```
barrier_struct {
    int K;
    spinlock s;
}
```

```
barrier_struct b;
b.init(K);
```

number of threads!

```
barrier (barrier_struct *b) {
    spinlock(b->s);
    b->k = b->k - 1;
    while (b->k > 0)
        sleep(b, b->s)
    wakeup(b);
    spinunlock(b->s);
}
```

waiting on this condition



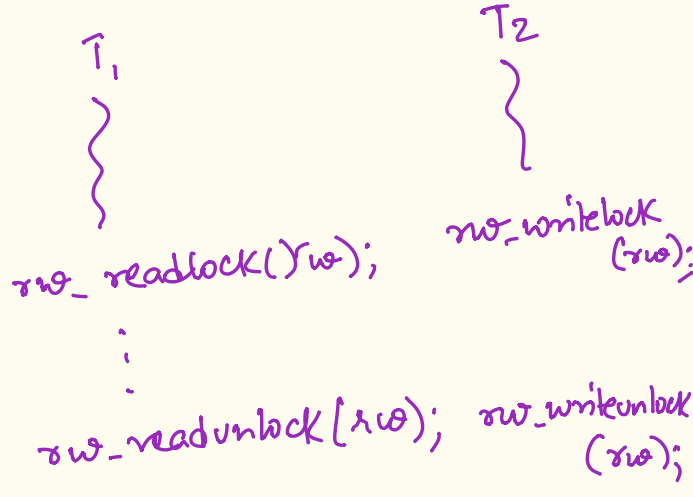
ReaderWriterlock rw;

(*) reader-writer lock.

- nreaders > 0 — fine
- nreaders > 0 — no writers
- nwriters = 1 — no writers or readers

* rw_readlock
* rw_readunlock
* rw_writelock
* rw_writeunlock

```
struct ReaderWriterlock {
    int nreaders;
    int nwriters;
    spinlock s;
}
```



```
rw_readlock (struct ReadWriteLock *rw)
{
    spinlock (rw->s);
    while (nwriters > 0)
        sleep (&rw->nwriters, rw->s);
    rw->nreaders ++;
    spinunlock (rw->s);
}
```

```
rw_writelock (*rw)
{
    spinlock (rw->s);
    while (nreaders > 0 or nwriters > 0)
        sleep (&rw->nreaders, m->s);
    rw->nwriters ++;
    spinunlock (rw->s);
}
```

H.W1 — unlock calls!

H.W2 — implementation of semaphore actions up & down using condition variables.

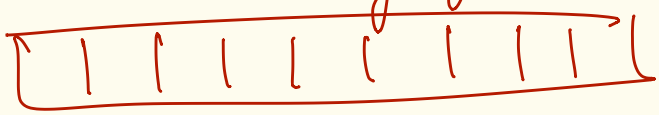
s → semaphore.

s.init (value); — initializes semaphore

up(s); — atomic increment

down(s); — atomic decrement — if (s < 0) sleep

produce (all of threads)



consumer

ready queue!

MAX size of queue.

if queue size == 0 → no consumption

if queue size == MAX → no addition.

scheduler()

semaphore S_1; S_2; S_3;

S_3.init(1); S_1.init(MAX); S_2.init(0);

```
producer {
    down(S1);
    down(S3);
    i = find index();
    up(S3);
    produce at i;
    up(S2);
}
```

unique loc^n in array → i

```
consumer {
    down(S2);
    down(S3);
    i = find locn();
    up(S3)
    consume at i;
    up(S1);
}
```

H.W

find out about (names / API & possible description)

7 pthread API calls!