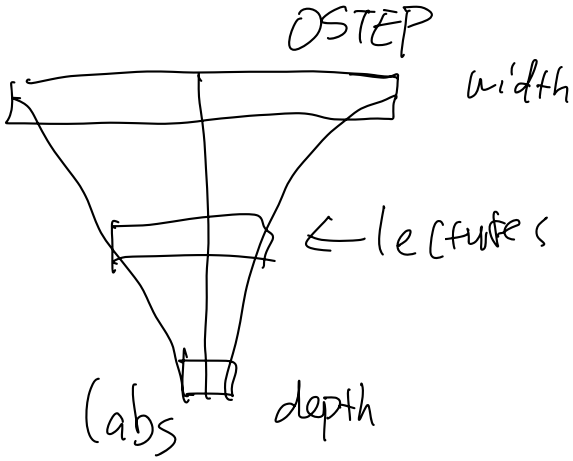
Week 6.b
CS3650
02/14 2024
https://naizhengtan.github.io/24spring/

1. Managing concurrency
2. Mutexes
3. Condition variables

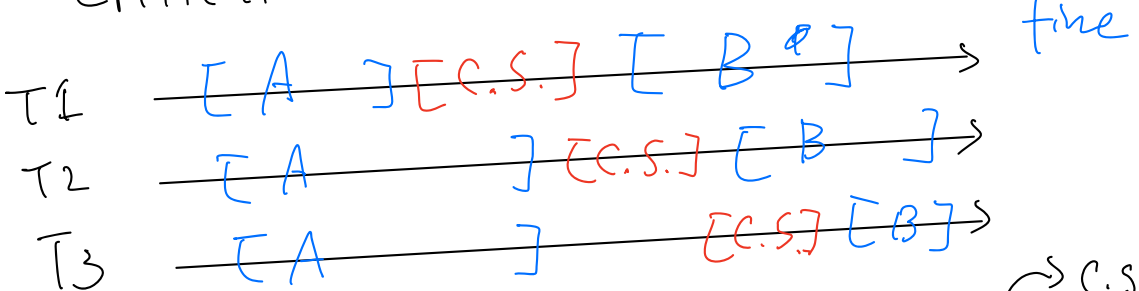
Admin:

- Survey
- readings & website
- labs & classes

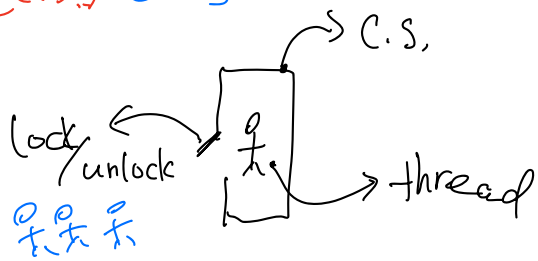


1. managing Concurrency

- Critical section



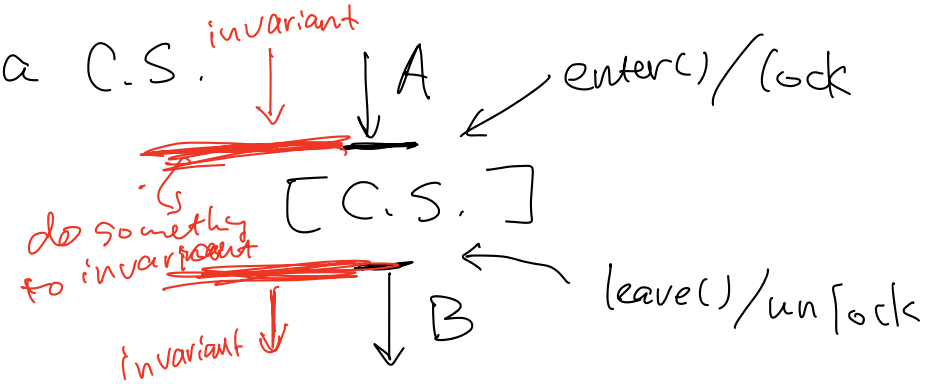
{A; C.S.; B}

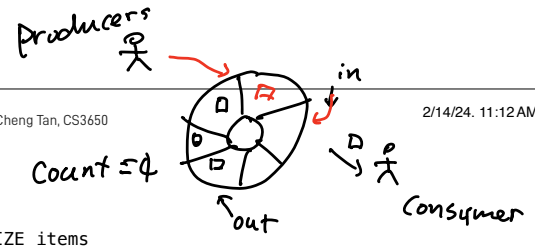


properties:

- mutual exclusion (atomicity)
- progress (no lockout)
- bounded wait (no deadlock)

- ~~protect~~ protect a C.S.

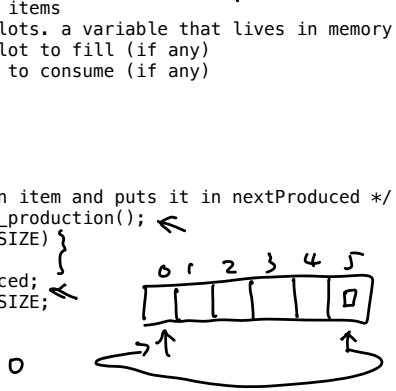




Week 6.b

1. Producer/consumer example:

```
    /*
        "buffer" stores BUFFER_SIZE items
        "count" is number of used slots. a variable that lives in memory
        "in" is next empty buffer slot to fill (if any)
        "out" is oldest filled slot to consume (if any)
    */

void producer (void *ignored) {

    for (;;) {
        /* next line produces an item and puts it in nextProduced */
        nextProduced = means_of_production();
        while (count == BUFFER_SIZE)
            ; // do nothing
        buffer [in] = nextProduced;
        in = (in + 1) % BUFFER_SIZE;
        count++;
    }
}

void consumer (void *ignored) {
    for (;;) {
        while (count == 0)
            ; // do nothing
        nextConsumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
        count--;
        /* next line abstractly consumes the item */
        consume_item(nextConsumed);
    }
}

    /*
        what count++ probably compiles to:
            reg1 <-- count # load
            reg1 <-- reg1 + 1 # increment register
            count <-- reg1 # store

        what count-- could compile to:
            reg2 <-- count # load
            reg2 <-- reg2 - 1 # decrement register
            count <-- reg2 # store
    */

    What happens if we get the following interleaving?

    T1  reg1 <-- count
    T1  reg1 <-- reg1 + 1
    T2  reg2 <-- count
    T2  reg2 <-- reg2 - 1
    T2  count <-- reg1
    T1  count <-- reg2
```



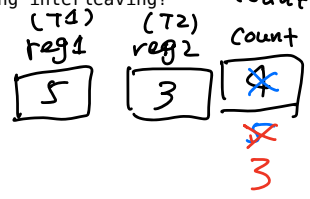
count = count + 1

- invariant

count == #items

violate

T1 : producer() count++;
T2 : consumer() count--;



2. Producer/consumer revisited [also known as bounded buffer]

2a. Producer/consumer [bounded buffer] with mutexes

```
    Mutex mutex;

void producer (void *ignored) {
    for (;;) {
        /* next line produces an item and puts it in nextProduced */
        nextProduced = means_of_production();

        acquire(&mutex);
        while (count == BUFFER_SIZE) {
            release(&mutex);
            yield(); /* or schedule() */
            acquire(&mutex);
        }

        buffer [in] = nextProduced;
        in = (in + 1) % BUFFER_SIZE;
        count++;
        release(&mutex);
    }
}

void consumer (void *ignored) {
    for (;;) {

        acquire(&mutex);
        while (count == 0) {
            release(&mutex);
            yield(); /* or schedule() */
            acquire(&mutex);
        }

        nextConsumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
        count--;
        release(&mutex);

        /* next line abstractly consumes the item */
        consume_item(nextConsumed);
    }
}
```

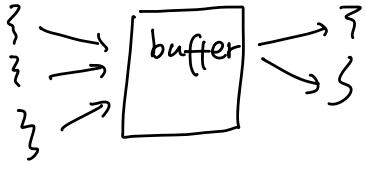
C.S.

Count == #items

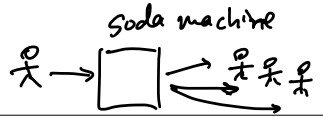
101 threads

busy waiting

1%



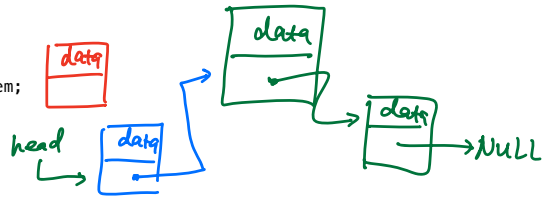
pipe

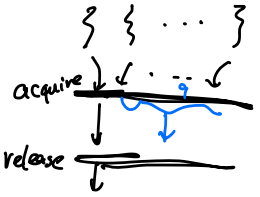


```
CS5600 Week06.a

1. Some concurrent programs. What is the point of these?

    [From S.V. Adve and K. Gharachorloo, IEEE Computer, December 1996,
    66-76. http://sadve.cs.illinois.edu/Publications/computer96.pdf]

    a. Can both "critical sections" run?

        int flag1 = 0, flag2 = 0;

        int main () {
            tid id = thread_create (p1, NULL);
            p2 (); thread_join (id);
        }

        void p1 (void *ignored) {
            flag1 = 1;
            if (!flag2) {
                critical_section_1 ();
            }
        }

        void p2 (void *ignored) {
            flag2 = 1;
            if (!flag1) {
                critical_section_2 ();
            }
        }

    b. Can use() be called with value 0, if p2 and p1 run concurrently?

        int data = 0, ready = 0;

        void p1 () {
            data = 2000;
            ready = 1;
        }
        int p2 () {
            while (!ready) {}
            use(data);
        }

    c. Can use() be called with value 0?

        int a = 0, b = 0;

        void p1 (void *ignored) { a = 1; }

        void p2 (void *ignored) {
          if (a == 1)
            b = 1;
        }

        void p3 (void *ignored) {
          if (b == 1)
            use (a);
        }
```



```
2. Protecting the linked list...

    Mutex list_mutex;

    insert(int data) {
      List_elem* l = new List_elem;
      l->data = data;

      acquire(&list_mutex);
C.S.  l->next = head;
      head = l;
      release(&list_mutex);
    }
```





2. Mutex

```
mutex_t m;
mutex_init(mutex_t *);
mutex_acquire(mutex_t *);
mutex_release(mutex_t *);
```

```
103    2b. Producer/consumer [bounded buffer] with mutexes and condition variables
104
105        Mutex mutex;
106        Cond nonempty;
107        Cond nonfull;
108
109        void producer (void *ignored) {
110          for (;;) {
111            /* next line produces an item and puts it in nextProduced */
112            nextProduced = means_of_production();
113
114            acquire(&mutex);
115            while (count == BUFFER_SIZE)
116              cond_wait(&nonfull, &mutex);
117
118            buffer [in] = nextProduced;
119            in = (in + 1) % BUFFER_SIZE;
120            count++;
121            cond_signal(&nonempty);
122            release(&mutex);
123          }
124        }
125
126        void consumer (void *ignored) {
127          for (;;) {
128
129            acquire(&mutex);
130            while (count == 0)
131              cond_wait(&nonempty, &mutex);
132
133            nextConsumed = buffer[out];
134            out = (out + 1) % BUFFER_SIZE;
135            count--;
136            cond_signal(&nonfull);
137            release(&mutex);
138
139            /* next line abstractly consumes the item */
140            consume_item(nextConsumed);
141          }
142        }
143
144
145    Question: why does cond_wait need to both release the mutex and
146    sleep? Why not:
147
148        while (count == BUFFER_SIZE) {
149          release(&mutex);
150          cond_wait(&nonfull);
151          acquire(&mutex);
152        }
153
154
```

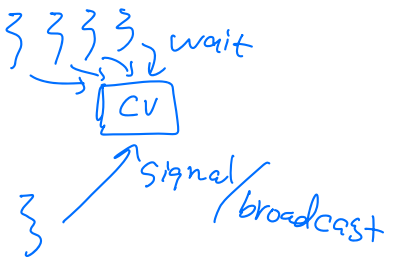
consumer
CV
signal
producer
C.S.
① unlock "m"; sleep on "c" until signaled
② re-acquire "m"; resume executing
(99)
wait HERE

Synchronization
- mutual exclusion ← Mutex
- scheduling constraints
  - condition variable

Condition variable interfaces:

Cond cv;

```
void cond_init (Cond *, ...);

void cond_wait(Cond *c, Mutex* m);

void cond_signal(Cond* c);

void cond_broadcast(Cond* c);
```



① unlock "m"; sleep on "c" until signaled

② re-acquire "m"; resume executing