Today (3/13)

-> 1 Handout @ Front

-> use handout from last time (first)

Concurrency: Primitives

- -> Locks [Data Races]
- -> Condition variables [Control Races]
  - => wait, signal

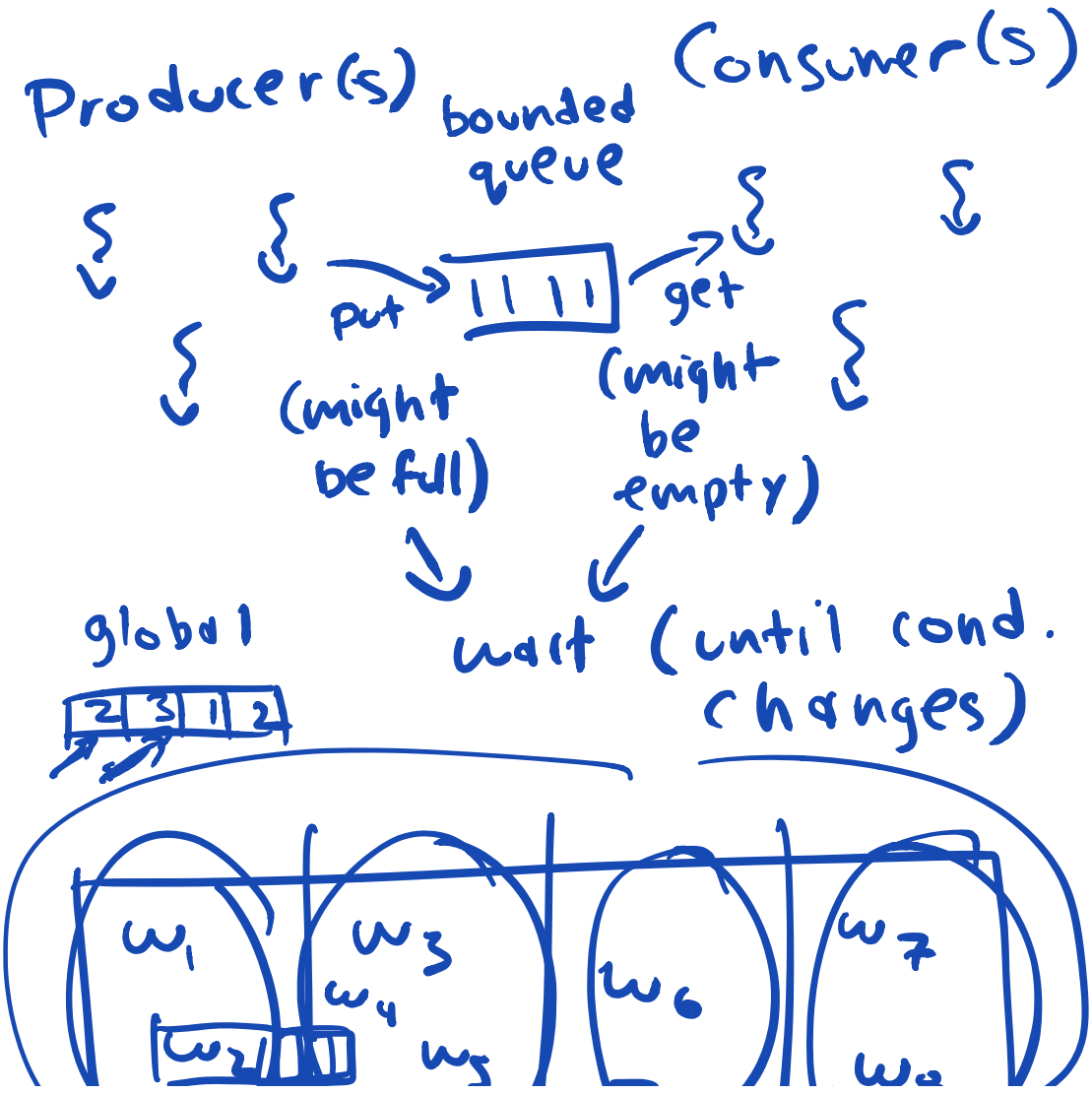
What's Left?

- => CV: Producer/Consumer (useful in pzip)
- => Semaphore: Historical

Next time:
[Deadlock]

Then: midterm

Producer/Consumer:

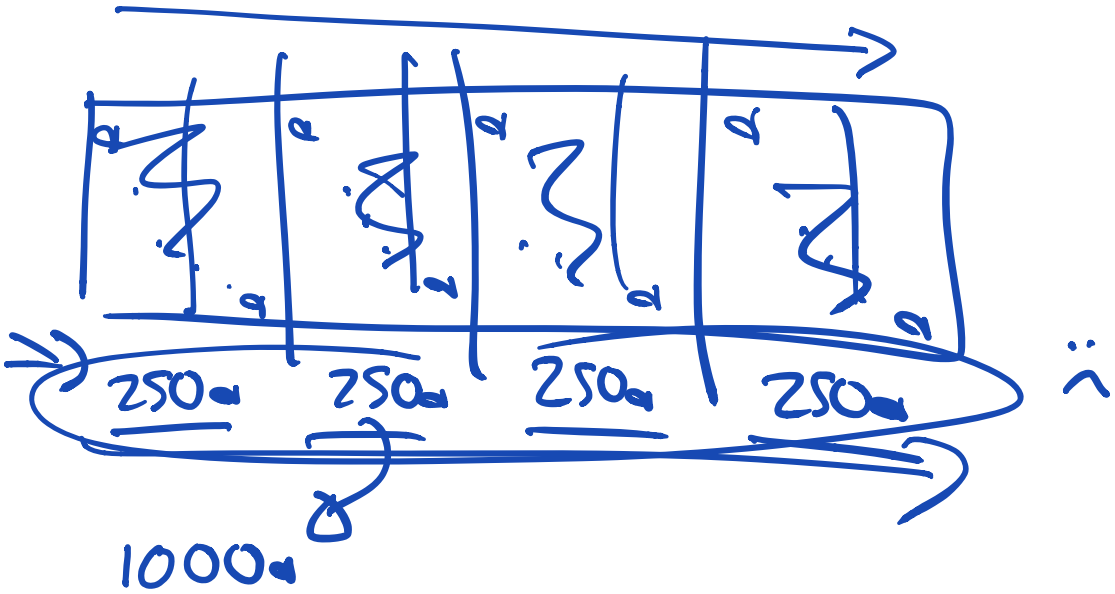


2 ⇒ 8

after 3 1 2

→ wait (join)

main: add up the results



Find a problem w/ solution 2:

P $P_1 P_2 P_4 P_5 P_6$

→ C_A C_1 C_2 C_3 ~~blocked / (sleeping)~~ → runnable

C_B (get) | C_1 C_2 C_4 C_5 C_6 |

numfull 0 0 0 0 0 ① 1 → 0

when C_A runs:
calls do-get()
w/ empty buffer

when wake up:
condition might have changed from the time when signal took place

=> use while to recheck condition

Admin:

=> contest winners (congrats!)

=> midterm:

covers a lot

(virt, concurrency)

next week: review

chapters: [1 ... 32]

midterm:

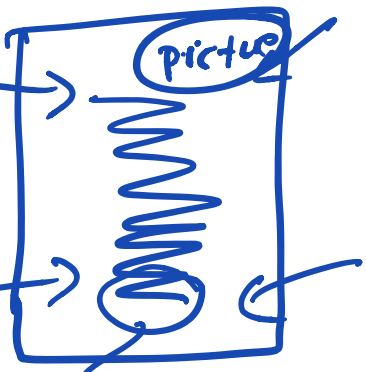
march 22,

thurs @

7:15pm -> done, dead, or other (9:15pm)

=> pages 370

"light reading"



closed book,

Closed notes

but: both sides

1 page

"cheat sheet"

~~reference~~

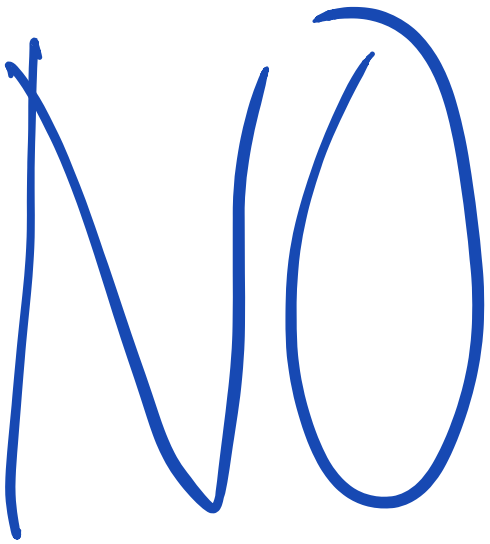
skip!

awesomeness negotiations:

[8½" x 11"]
~~feet~~
~~meters~~

hand written,
printed
laser etched,
....
stickers OK!

NO →



← no computers

format? => { practice exams yes?

Semaphore: (Dijkstra on OS construction "T.H.E.")

single replacement for both locks+CVs => concurrency

abstract object w/ two operations:

→ wait

→ post

semaphore has value (determines behavior)

could use

locks + CVs OR semaphores

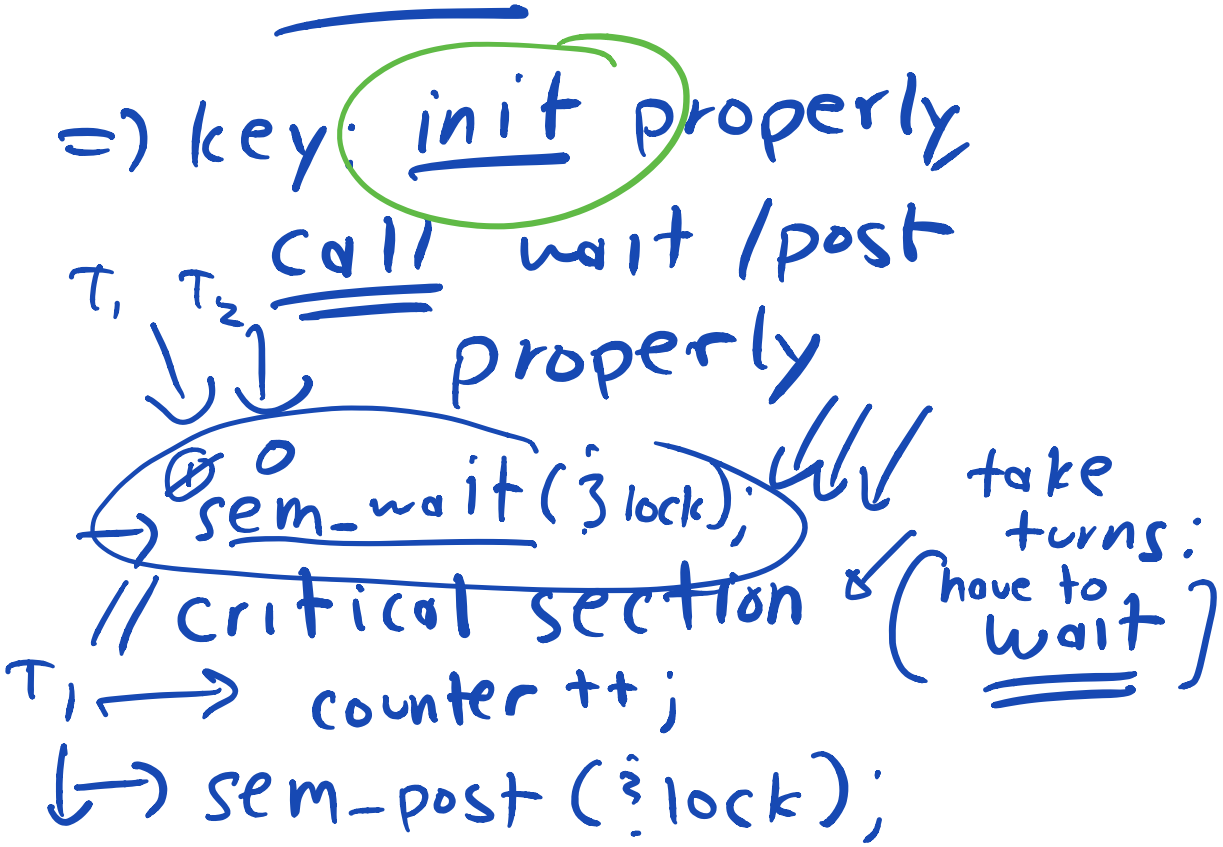
me

you?

use sema as lock

=> key: init properly

call wait/post properly



→ sem_init(&lock, 1);

Fork / join : sem_t cv;

```
child() {
  // do work
  sem_post(?);
}
```

```
parent() {
  sem_init(?cv, ? 0);
  thread_create(child);
  [sem_wait(?)]
}
```

=> yeah!

Semaphores
P(), V()

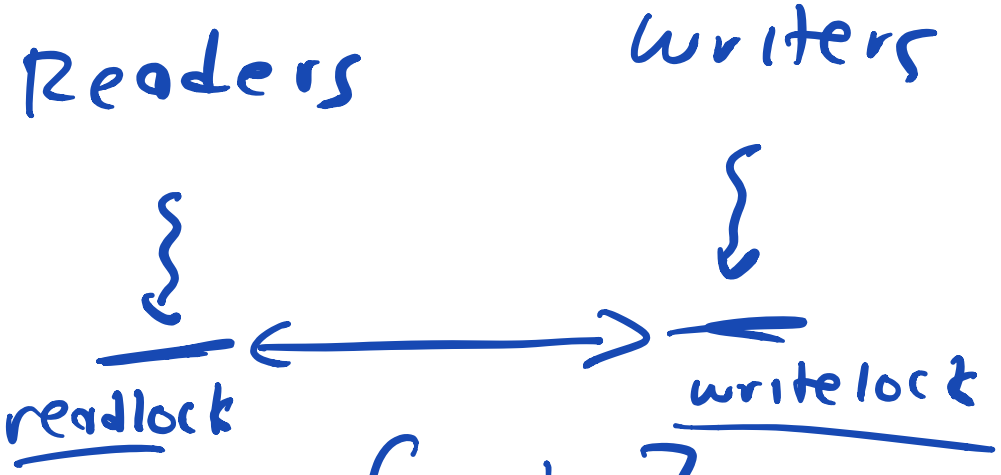
---

reader/writer locks :

=> many readers can be in crit. section at once

OR

one writer



Problem: Starvation [writer]

Projects: Open Questions

- Time benefits of zipping while writing?
- pzip design pattern
- How to avoid fgetc()?

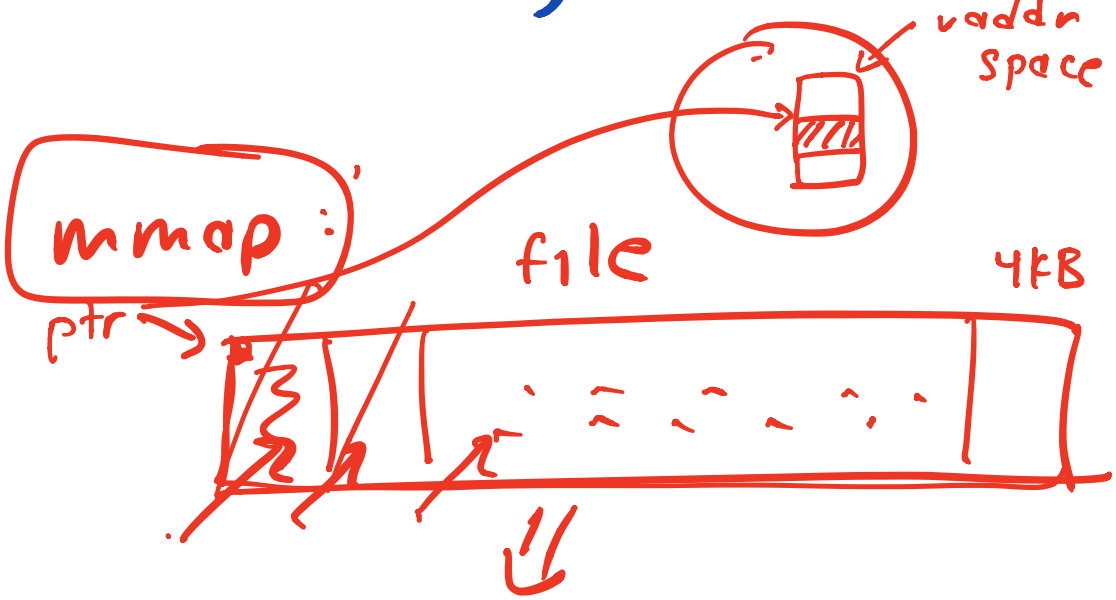
→ ~~Assume files fit into memory?~~

→ Zip: any characters excluded?

→ Optimal write buffer size?

part b:

→ accessing protection bits



ptr = mmap (....);

faults in pages of file lazily

char *

ptr -> file

single threaded:

long long int

```
for (i=0;i<size;i++)
    *ptr // deref

    ptr++;
}
```

write buffer size?

write(fd, buffer, size)

system call overhead

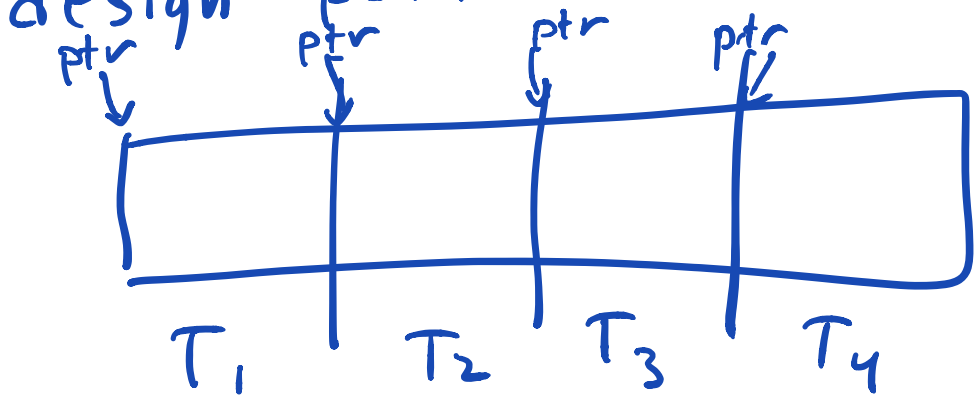
if very small:

=) how big?

be empirical

try 4KB, 64KB, ...

design pattern:



simple:

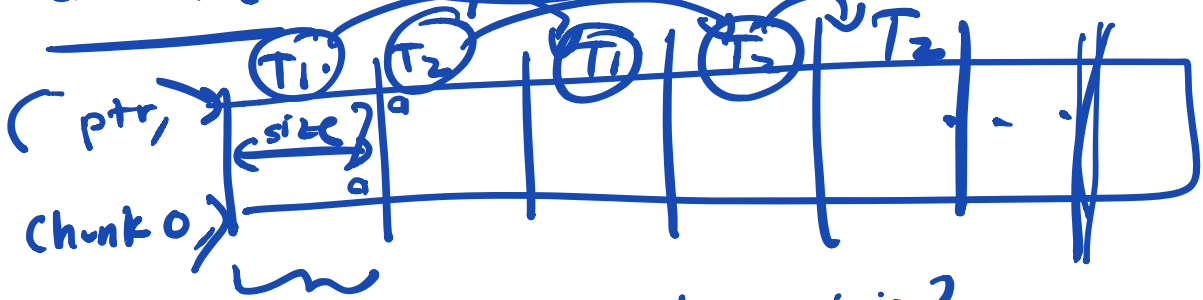
create threads

assign static work to each

wait to zip each piece

stitch together final results

advanced: dynamic



chunk size: how big?

1 MB

main:

create descriptions

L ... wait

waiting for workers to be done

stitch together results

workers:

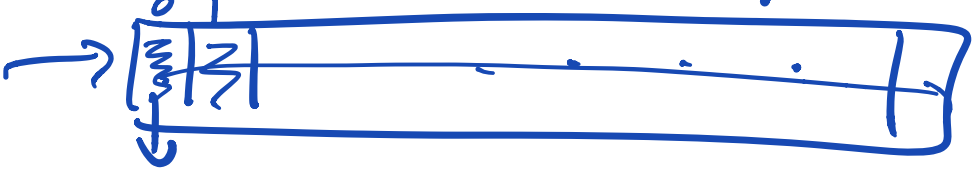
get a work desc,

do work: until signal: done

put result somewhere

return

output array: 1 entry per chunk



list

: of zipped

contents

3a → 4b → 16m → 2r