CS347m / CS219

Lecture 20

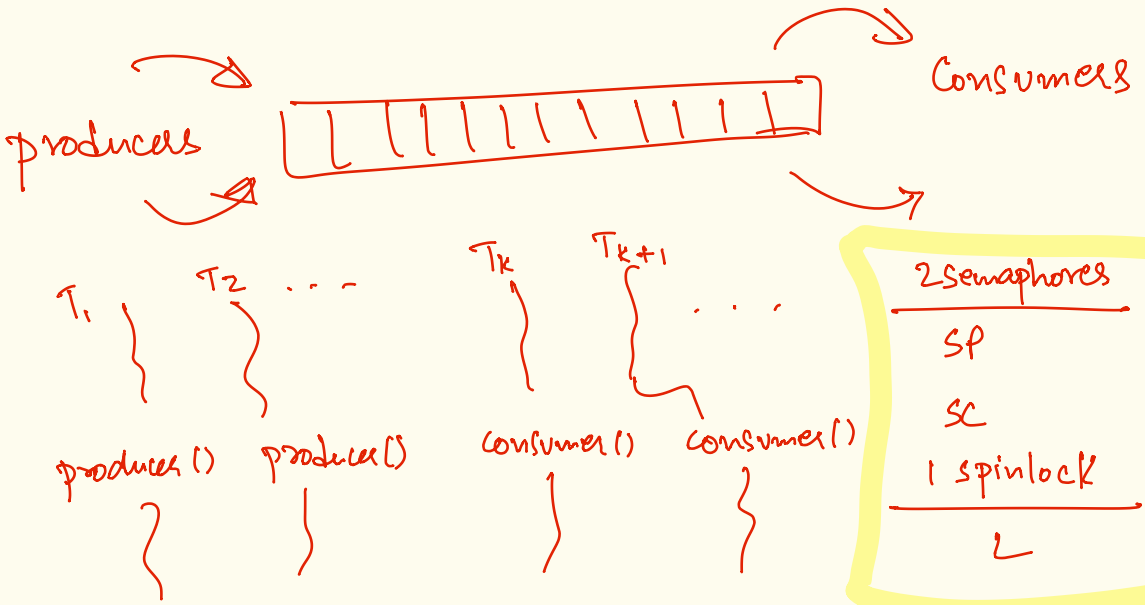
recap: concurrency & synchronization

spinlocks, mutex, semaphores, condition variables, barrier, dining philosphers problem, deadlock.

The Little Book of Semaphores ~ Allen D.

└ tons of interesting synchronization / concurrency related problem & solution descriptions.

#(1) producer & consumer problem.



2 semaphores
sp
sc
1 spinlock
L

Queue
MAX

producer only if qsize < MAX

consume only if qsize > 0

sp.init(MAX);
sc.init(0);

```
producer() {
    down(sp);
    spinlock(L);
        addtoqueue();
    spinunlock(L);
    up(sc);
}
```

```
consumer() {
    down(sc);
    spinlock(L);
        dequeue();
    spinunlock(L);
    up(sp);
}
```

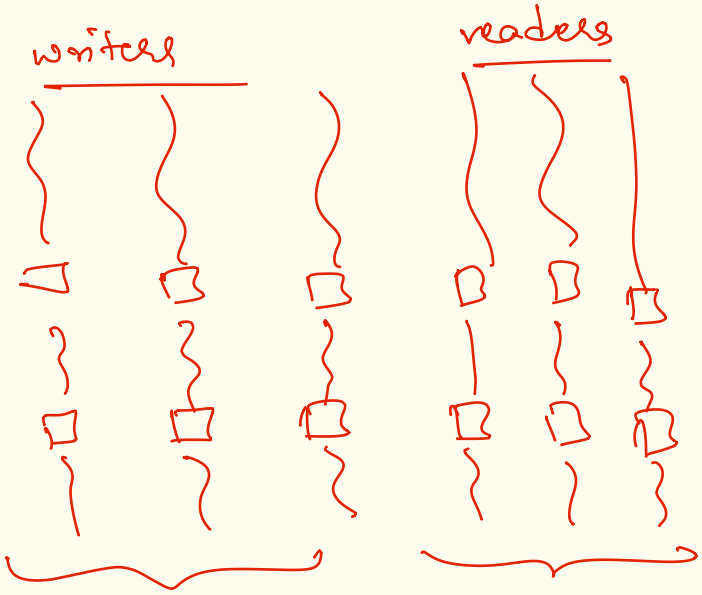
H.W — producer-consumer & barrier implementation using condition variables.

readers-writers problem (implementation using condition variables)

~ a shared region/object & accessed concurrently by reader or writer threads

reader → read value

writer → writes/updates the value



synchronization constraints:

(i) many readers — okay. (accessing shared object simultaneously)

(ii) (concurrent) many writers — not okay.

(iii) if reader, NO writer

(iii) if writer, NO reader, NO writer

H.W — re-implement RW locks using semaphores.

(*) Solution –

```
spinlock L;          ——— lock to access shared objects.
int nwriters = 0;
int nreaders = 0;
```

```
rw_readlock() {
    spinlock(L);
    while (nwriters > 0)
        sleep(&nwriters, L);
    nreaders++;
    spinunlock(L);
    // do read
}
```

condition → nwriters; value/variable

```
rw_readunlock() {
    spinlock(L);
    nreaders--;
    if (nreaders == 0)
        wakeup(&nreaders);
    spinunlock(L);
}
```

```
rw_writelock() {
    spinlock(L);
    while (nreaders > 0 OR
           nwriters > 0)
        sleep(&nreaders, L);
    nwriters++;
    spinunlock(L);
    // do write
}
```

```
rw_writeunlock() {
    spinlock(L);
    nwriters--;
    wakeup(&nwriters);
    wakeup(&nreaders);
    Spinunlock(L);
}
```

persistence — not ephemeral!

- all (cpu) work depends on — instructions, data
  sequence / arrangement of bytes.

Q where to get these bytes from?

- naming — refer
- storage / area — objects are parked
- organization — search objects

persistent storage vs ephemeral storage (fleeting not permanent)

- HDD - hard disk
- device controller & device IO interface
- needs a device driver
- persistent
- slower
- very large capacity

- main memory / RAM
- volatile / ephemeral
- address bus interface
- fast
- small/medium capacity

---

the file abstraction (the OS abstraction for users to access persistent storage)

- linear / contiguous sequence of bytes. } organization of data
- persistent.
- properties: name, size, timestamps, access control, links — PATH

abstraction is provided by the file system

FS interface: open, close, seek, read, write, truncate

```
int fd = open("worldpeace.txt", ...);
```

- `fd`: integer. index in fd table, stored in PCB
- `"worldpeace.txt"`: path/filename
- `...`: mode

```
read(fd, buf, 1);
```