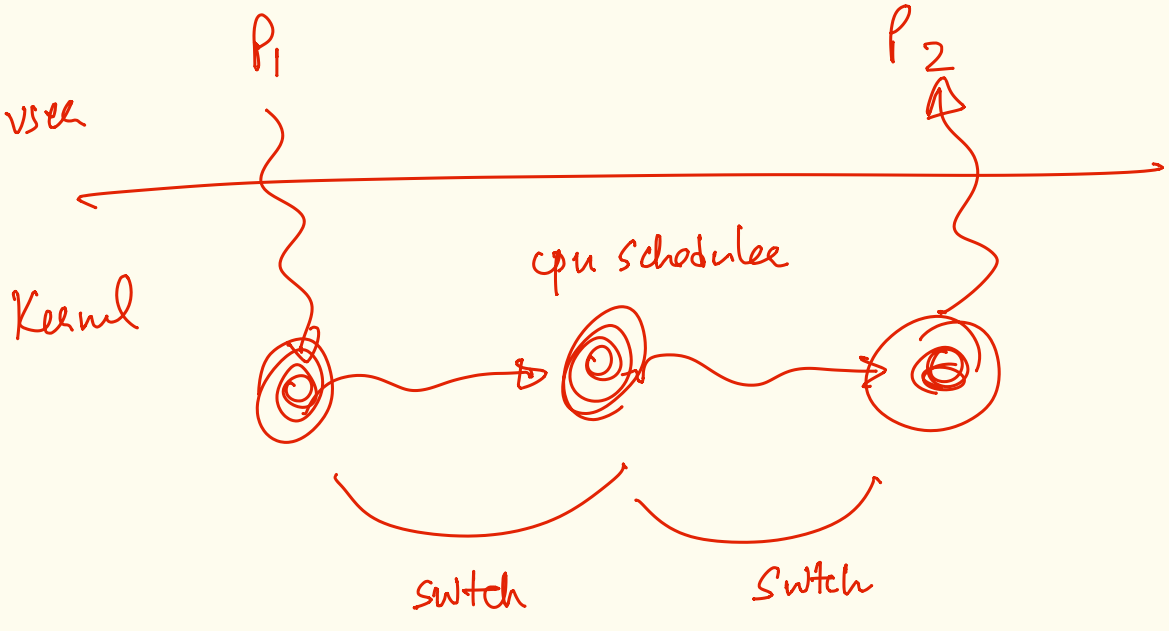
CS 219 Lecture 22 Lab 7 today afternoon.

* scheduling — mechanism — policy

① context save & restore
- process state (user context)
- Kernel state (Kernel context)

② s/wing mechanism

xv6: per-cpu scheduler / Kernel thread



policies

```
while (1) {
    np = get_next_process ();
    schedule (np);
}
```

get_next_process () → scheduling policy

schedule (np) → switching mechanism

categorization of scheduling policies.

1. work conserving vs non work conserving
2. pre-emptive vs non pre-emptive
3. real-time vs best-effort
   - deadlines, finish times, ...
     - periodicity ...
   - fair proportionate priority etc.
4. IO vs batch processes.

stepped differentiated RR

high pri

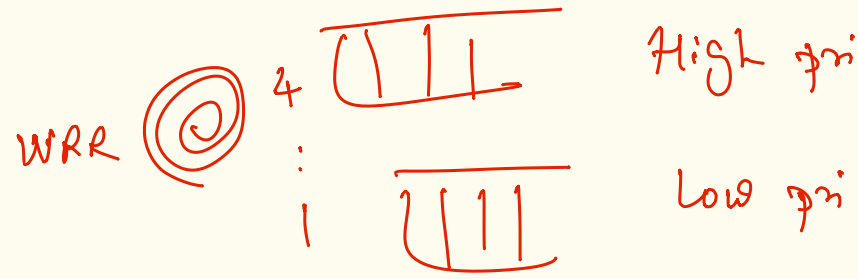
low pri

ready queues / run queues

high priority process always scheduled before low priority process.

scheduling policy invariant

- proportionate scheduler



---

best effort: RR, WRR, lottery scheduling, CFS

real time: EDF, SJF, EFT, ...

Synchronization & Concurrency

threads (of execution) / instances of execution

① ~ multiple processes executing on multiple CPUs simultaneously

② ~ multi-threading / multiple threads of a process on multiple CPUs

③ ~ multiple instances of the Kernel on multiple CPUs simultaneously.

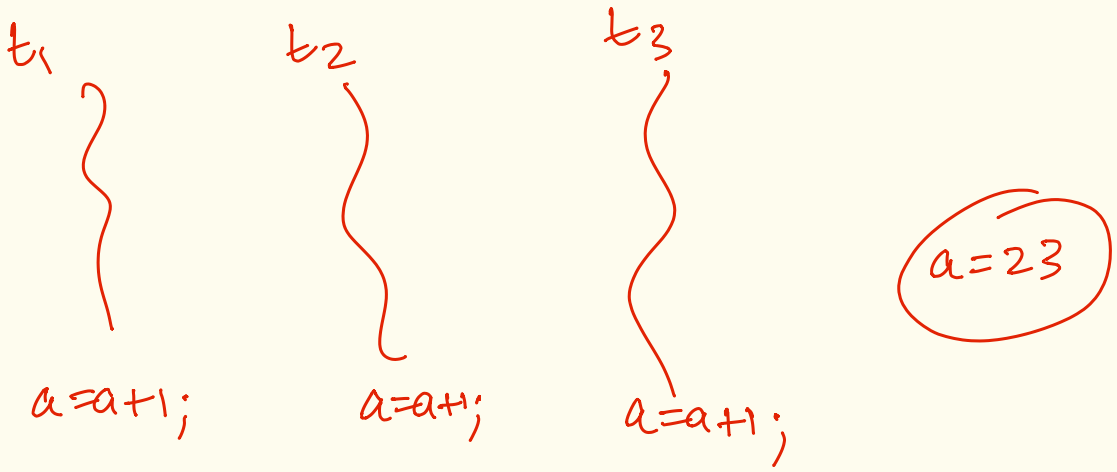
setup1 + setup2 ~ concurrency!

~ while in setup1, there exists shared memory regions / objects. (syscall linux: shmem)

① processes sharing part of the page table.

② w/ threads, all threads share process address space.

③ Kernel has a single address space (single entity)



Q what is value of 'a' after t_1, t_2 & t_3 execute the update instruction?

26, 25, 24

any of this is possible!

Q why?

Q how to prevent this non-determinism?