- (finish semaphore mpl)
- Monitor design pattern
- (quiz)
Using semaphores for thread control (per-thread sema)

To sleep current thread:

- allocate new semaphore \( s \)
- put \( s \) in a data structure
- \( P(s) \)
  - decrement

To wake up sleeping thread:

- get \( s \) from data structure
- \( V(s) \)
  - increment
- delete \( s \)
To handle a syscall (monolithic kernel)

1. Save state as if executing a normal function call

2. Perform the operation (e.g., communicate w/ device)
   - Branch (normal jump) to driver code to interact w/ device.
   (Note: if microkernel, schedule driver process)

3. Return from syscall (restoring state)
I/O bound

Many threads can wait simultaneously.

thread / proc creation overhead

sequential

overhead

processes

threaded

$n=1$  $n=2$  $n=3$  $n \to$  $n=100$
CPU bound threads/proc

Single core

Threads

I/O

Proc

Thread

multicore (4)
expected

multicore (actual)

only one CPU, so just overhead

more threads ⇒ more CPUs (if avail) ⇒ faster

just overhead
class Semaphore (object):
    def init (self, val):
        self.val = val  # protected by self.lock
        self.lock = 0  # TAB lock
        self.queue = []  # protected by self.lock
        # invariant: val >= 0, val = 0 if q is nonempty
    def P(self):
        while TAB(self.lock):
            yield()  # or pass
        if self.val > 0:
            self.val -=
            self.lock = 0
            return
        else:
            self.queue.enqueue(running_thread, TCB)
        release lock; deschedule(current thread)
        return

    def V(self):
        while TAB(lock):
            yield()  # or pass
        if queue.empty():
            self.val +=
            lock = 0
            return
        else:
            dequeue a thread t, mark t as ready
            lock = 0
            return
Monitor design pattern

- Invented by Sir Tony Hoare.
- Closely related to Hoare logic.
- Get support from PL.

Monitor object is like a normal OOP object,

- state (member vars)
- methods
- at any time, only one thread running inside a given monitor.
- able to wait for certain conditions to hold (e.g., wait for counter > 0)
Semaphore impl. as a monitor (Hoare style)

```python
class Semaphore(Monitor):
    def __init__(self, value):
        self.val = value  # invariant: value >= 0

    def P(self):
        wait until self.val > 0
        self.val -=

    def V(self):
        self.val +=
```

Hoare-style.
In practice, need a little more effort from programmers.
Bounded buffer monitor (Hoare style)

class Buffer (Monitor):
    def __init__(self):
        self.in = 0    # next empty slot
        self.out = 0   # next full slot (if any)
        self.buf = [Object[N]]
        # invariant: 0 ≤ in < N
        # in = out represents empty queue,
        # in +1 = out represents full queue

    def put(self, obj):
        wait until in +1 + out (mod N)
        buf[in ++] = obj

    def get(self, obj):
        wait until in = out
        return buf[in + out (mod N)]]