CSE 330: Operating Systems

Class: 16 **Date:** 3/1

Spring 2018

Note Title

Monitors ⟶ set of methods

- methods execute in critical sections
- wait (cond) → always blocks
- signal (cond) → wake up 1 thread if any ⟵ else NO-OP

producer - consumer

Count = 0 // init

conditions → prod, cons

```
produce
    { if (count == N)
            wait (prod)
      count++
      add item to buffer
      signal (cons)
    }
```

```
cons
    { if (count == 0)
            wait (cons)
      count --
      take item
      signal (prod)
    }
```

Readers/writers (starvation free) { rc, wc, rwc, wwc } = 0

read, write] condit

reader entry

{ if((wc>0)||(wwc>0)) [rwc++ wait(read) rwc--]
rc++
signal(read) ← wake up other readers if any
}

reader exit

{ rc--; if(rc==0) signal(write) }

writer entry

```
{ if((wc>0)||(rc>0)) { wwc++
                       wait(write)
                       wwc--
  wc++
}
```

writer exit

```
{ wc--
  if (rwc>0) signal(read)
  else signal(write) }
```

Dining Philosophers

philo → array of conditions available ....

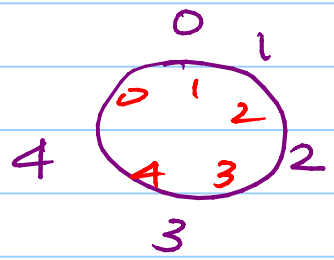
pickup
- while !(available left chopstic && available right chopstic)
  - wait (philo[i])
- make chopsticks unavailable

putdown
- make chopsticks available
- wake up neighbors

int chop[5] init → 1 ; — available

condition philo[5] ;

pickup(i) → [while !((chop[i] == 1) && (chop[(i+1)%5] == 1)) { wait(philo[i]) }
chop[i] = chop[(i+1)%5] = 0



putdown(i) → [chop[i] = chop[(i+1)%5] = 1
signal(philo[(i+4) mod 5])
signal(philo[(i+1) mod 5])

i-1

Monitors vs Semaphores

→ equivalent

① → implement semaphores with monitors

→ sem <= monitor

② reverse sem >= monitor

≡

implement sem with monitor

each sem → [counter, cond]

```
init(s, v) { s.count = v }

P(s) { s.count--
       if (s.count < 0) wait(s.cond) }

V(s) { s.count++
       if (s.count <= 0) { signal(s.cond) } }
```

```
Monitor M;   // the monitor

Entry func(..) {…

… ..

c.wait()

… ..

c.signal() ..

}

End Monitor;
```

```
semaphore mutex, next, x_sem;
int next_count, x_count;

func()
{
   P(mutex)  // entry section

   x_count ++    // "wait call"

   if (next_count > 0)  V(next)

   else V(mutex)

   P(x_sem)

   X_count--;
   if(x_count > 0) {  // "signal" called
      next_count++;
      V(x_sem)
      P(next)
      next_count--;
   }

   if (next_count > 0) V(next) // "exit" section
   else V(mutex)

}
```