CS4110 Announcements

- Tutorial on Semaphores (Lects 9/17, 9/19) for students who attended Tapa Conference.
  But any CS4110 student welcome to attend Sunday 9/29 3pm-4:30pm Hollister B14 (?)

- Assignment 2 hints
  * For problem 1:
    - Critical sections always terminate.
    - Non-critical sections (NCS) is not required to terminate and might not!

  * For problem 2:
    - Mutual exclusion protocols are expected to work for \( N \geq 2 \) processes.

- Looking ahead: Prelim 1 is Thurs 10/10 + 2 weeks 😞
  Likely review session 10/8 in class
Beyond Semaphores

Synchronization primitives
- Shared memory
- Semaphores
- Monitors & Regions

Tension:
- Generality / expressiveness
- Usability / clarity

Exploit:
- Syntax
- Compiler (= syntax checking)
- Run-time
schematic representation of monitor
Support for synchronization =

```
mutex + Cond Sych
```

per operation

```
op. operation (p1, p2 ...)
  var locals
  
  Require that B holds (*
  
  end
```
BB: Monitor

var buff[0..N] of portion
stht: 0..N
len: 0..N+1
slots: 0..N+1

insert: operation (v: portion)
need a slot!
buff[stht+len] := v
len := len + 1; slots := slots - 1

rmv: operation (v: val: portion)
need a portion!
val := buff[stht]
len := len - 1; slots := slots + 1
stht := stht + 1 mod N+1

begin
stht := 0
len := 0
slots := 0
end
Condition variables

var C: conditon $\rightarrow$ associate $B_0$

$\triangleright$ wait
1. releases mutual lock for monitor
2. blocks process

$\triangleright$ continue: exit monitor & unblock I process
BB: 

```plaintext
var buff [0..N] of portion

start: 0..N
len: 0..N+1
slots: 0..N+1

space = Cond 1
stuff = Cond 2

if slots > 0 then
  B_space
  B_stuff

insert: operation (var val: portion)

need a slot!
if slots = 0 then space.out

buff [start+len] := val
len := len + 1; slots := slots - 1
stuff.continue

rmv: operation (var val: portion)

need a portion!
if len = 0 then stuff.wait

val := buff [start]
len := len - 1; slots := slots + 1
start := start + 1 mod N+1
space.continue

begin
  start := 0
  len := 0
  slots := 0
end BB
```
Suppose signal operation does not cause output?

If A runs after C.signal

\[ \text{\texttt{C.wait}} \rightarrow \text{\texttt{C.wait}} \rightarrow \text{\texttt{C.wait}} \rightarrow \text{\texttt{C.wait}} \]

Then C.wait \texttt{signal} urgent semantics is: before new entry to monitor.

Suppose signal operation does not cause output?

\[ \text{\texttt{C.wait}} \rightarrow \text{\texttt{C.wait}} \rightarrow \text{\texttt{C.wait}} \rightarrow \text{\texttt{C.wait}} \]
Implemention of signal/urgent semanties

- entry queue for monitor
- condition queue for each condition variable
- urgent queue for monitor

Monitor call: If monitor in use
  then add thread to entry queue
  else grant access

C.wait: Put thread on queue for c
  Invoke scheduler

C.signal: Put thread on urgent queue
  If condition queue for c not empty
    then run head next
  else invoke scheduler

Scheduler: Any thread on urgent queue?
  Run it!
Else any thread on entry queue?
  Run it!
Implementation of signal/urgent semaphores with semaphores

```
m: monitor
{
  op: operation(---)
  |
  |
  c: wait
  |
  |
  c: signal
}

var mubex: semaphore init(1)
semc: semaphore init(0)
urmg: semaphore init(0)
```

```
C.count := C.count + 1
if urgcount > 0
  then urgcount := urgcount - 1
  V(urmg)
else
  V(mubex)
```

```
P(mubex)
```

```
P(semc)
```

```
if urgcount > 0
  then urgcount := urgcount - 1
  V(semc)
P(urmg)
```

```
if C.count > 0
  then C.count := C.count - 1
  V(semc)
P(urmg)
```

```
if urgcount > 0
  then V(urmg)
else V(mubex)
```
Acut: monitor

var ant: integer

inc: condition

if ant > m then inc: ant := ant

end

deposit: operator (n: integer)

ant := ant + n

inc: signal

end

withdraw: operator (w: integer)

if ant <= w then inc: wait

ant := ant - w

end

end Acut