6.033 Spring 2008

Lecture 7

Threads

Remember: Send with Locking

```
send(p, m):
  while true:
    acquire(p.lock)
    if p.in - p.out < N:
      p.buffer[p.in mod N] ← m
      p.in ← p.in + 1
      release(p.lock)
      return
    release(p.lock)
```

Send with Yield

```
send(p, m):
  while true:
    acquire(p.lock)
    if p.in - p.out < N:
      p.buffer[p.in mod N] ← m
      p.in ← p.in + 1
      release(p.lock)
      return
    release(p.lock)
    yield()
```

Send and Receive

```
send(p, m):
  while true:
    if something to do: do it
    else: yield()

receive(p):
  while true:
    if something to do: do it
    else: yield()
```

```
yield():
  acquire(tt_lock)
  id = cpu_table[CPU()].thread
  ttable[id].state = RUNNABLE
  ttable[id].sp = SP                    } save

  do:
    id = (id + 1) mod N
  while ttable[id].state != RUNNABLE

  ttable[id].state = RUNNING
  SP = ttable[id].sp
  cpu_table[CPU()].thread = id          } restore
  release(tt_lock)
```

Coping w/ exiting threads (1)

```
yield():
  acquire(tt_lock)
  id = cpu_table[CPU()].thread
  if ttable[id].state != EXITING:
    ttable[id].state = RUNNABLE
    ttable[id].sp = SP

  do:                                   } idle loop
    id = (id + 1) mod N
  while ttable[id].state != RUNNABLE

  ttable[id].state = RUNNING
  SP = ttable[id].sp
  cpu_table[CPU()].thread = id
  release(tt_lock)
```

Coping w/ exiting threads (2)

```
yield():
   acquire(tt_lock)
   id = cpu_table[CPU()].thread
   if ttable[id].state != EXITING:
      ttable[id].state = RUNNABLE
      ttable[id].sp = SP

   do:                                          } idle
      id = (id + 1) mod N                       } loop
      release(tt_lock); acquire(tt_lock)
   while ttable[id].state != RUNNABLE

   ttable[id].state = RUNNING
   SP = ttable[id].sp
   cpu_table[CPU()].thread = id
   release(tt_lock)
```

Coping w/ exiting threads (3)

```
yield():
   acquire(tt_lock)
   id = cpu_table[CPU()].thread
   if ttable[id].state != EXITING:
      ttable[id].state = RUNNABLE
      ttable[id].sp = SP
   SP = cpu_table[CPU()].sp

   do:
      id = (id + 1) mod N
      release(tt_lock); acquire(tt_lock)
   while ttable[id].state != RUNNABLE

   ttable[id].state = RUNNING
   SP = ttable[id].sp
   cpu_table[CPU()].thread = id
   release(tt_lock)
```

Remember: Send with Yield

```
send(p, m):
   while true:
      acquire(p.lock)
      if p.in - p.out < N:
         p.buffer[p.in mod N] ← m
         p.in ← p.in + 1
         release(p.lock)
         return
      release(p.lock)
      yield()
```

This doesn't work.

```
send(p, m):
   while true:
      acquire(p.lock)
      if p.in - p.out < N:
         p.buffer[p.in mod N] ← m
         p.in ← p.in + 1
         notify(p.notempty)
         release(p.lock)
         return
      release(p.lock)
      wait(p.notfull)
```

This works.

```
send(p, m):
   acquire(p.lock)
   while p.in - p.out == N:
      wait(p.notfull, p.lock)
   p.buffer[p.in mod N] ← m
   p.in ← p.in + 1
   notify(p.notempty)
   release(p.lock)
```

```
wait(cvar, lock):
   acquire(tt_lock)
   release(lock)
   ttable[id].cvar = cvar
   ttable[id].state = WAITING
   yield_tt_lock_already_held()
   acquire(lock)
```

```
wait(cvar, lock):
  acquire(tt_lock)
  release(lock)
  ttable[id].cvar = cvar
  ttable[id].state = WAITING
  yield_tt_lock_already_held()
  acquire(lock)

notify(cvar):
  acquire(tt_lock)
  for i = 0 to N-1:
    if ttable[I].cvar == cvar and
        ttable[I].state == WAITING:
      ttable[I].state = RUNNABLE
  release(tt_lock)
```