L7: Threads

Nickolai Zeldovich
6.033 Spring 2012
Recall: send with locking

send(bb, m):
    acquire(bb.send_lock)
    while True:
        if bb.in - bb.out < N:
            bb.buf[bb.in mod N] ← m
            bb.in ← bb.in + 1
    release(bb.send_lock)
    return
Send and receive with yield

send(bb, m):
    acquire(bb.lock)
    while True:
        if bb.in – bb.out < N: ...
        release(bb.lock)
        yield()
        acquire(bb.lock)

receive(bb):
    acquire(bb.lock)
    while True:
        if bb.in > bb.out: ...
        release(bb.lock)
        yield()
        acquire(bb.lock)
yield():
    acquire(t_lock)
    id = cpus[CPU].thread
    threads[id].state = RUNNABLE
    threads[id].sp = SP

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

threads[id].state = RUNNING
SP = threads[id].sp
cpus[CPU].thread = id
release(t_lock)
yield():
    acquire(t_lock)
    id = cpus[CPU].thread
    threads[id].state = RUNNABLE
    threads[id].sp = SP

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

    threads[id].state = RUNNING
    SP = threads[id].sp
    cpus[CPU].thread = id
    release(t_lock)

    suspend current thread
yield():
    acquire(t_lock)
    id = cpus[CPU].thread
    threads[id].state = RUNNABLE
    threads[id].sp = SP

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

    threads[id].state = RUNNING
    SP = threads[id].sp
    cpus[CPU].thread = id
    release(t_lock)

} suspend current thread

} choose new thread
yield():
    acquire(t_lock)
    id = cpus[CPU].thread
    threads[id].state = RUNNABLE
    threads[id].sp = SP

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

    threads[id].state = RUNNING
    SP = threads[id].sp
    cpus[CPU].thread = id

    release(t_lock)

    suspend current thread

    choose new thread

    resume new thread

    new thread
Send with yield, again

send(bb, m):
    acquire(bb.lock)
    while True:
        if bb.in – bb.out < N:
            bb.buf[bb.in mod N] ← m
            bb.in ← bb.in + 1
            release(bb.lock)
            return
    release(bb.lock)
yield()
    acquire(bb.lock)
Send with wait / notify

send(bb, m):
    acquire(bb.lock)
    while True:
        if bb.in – bb.out < N:
            bb.buf[bb.in mod N] ← m
            bb.in ← bb.in + 1
            release(bb.lock)
            notify(bb.empty)
            return
    release(bb.lock)
    yield()
    acquire(bb.lock)
    wait(bb.full, bb.lock)
Wait and notify

```
wait(cvar, lock):
    acquire(t_lock)
    release(lock)
    threads[id].cvar = cvar
    threads[id].state = WAITING
    yield_wait()  # will be a little different than yield
    release(t_lock)
    acquire(lock)
```
Wait and notify

wait(cvar, lock):
    acquire(t_lock)
    release(lock)
    threads[id].cvar = cvar
    threads[id].state = WAITING
    yield_wait()  # will be a little different than yield
    release(t_lock)
    acquire(lock)

notify(cvar):
    acquire(t_lock)
    for i = 0 to N-1:
        if threads[i].cvar == cvar && threads[i].state == WAITING:
            threads[i].state = RUNNABLE
    release(t_lock)
Recall: original yield

yield():
  acquire(t_lock)
  id = cpus[CPU].thread
  threads[id].state = RUNNABLE
  threads[id].sp = SP

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

  threads[id].state = RUNNING
  SP = threads[id].sp
  cpus[CPU].thread = id
  release(t_lock)

  suspend current thread

  choose new thread

  resume new thread
Yield for wait, first attempt

```python
def yield_wait():
    acquire(t_lock)
    id = cpus[CPU].thread
    threads[id].state = RUNNING
    threads[id].sp = SP

    while threads[id].state != Runnable:
        id = (id + 1) % N

    threads[id].state = RUNNING
    SP = threads[id].sp
    cpus[CPU].thread = id
    release(t_lock)
```
Yield for wait

```python
yield_wait():
    id = cpus[CPU].thread
    threads[id].sp = SP
    SP = cpus[CPU].stack

    do:
        id = (id + 1) mod N
        release(t_lock)
        acquire(t_lock)
    while threads[id].state ≠ Runnable

    threads[id].state = RUNNING
    SP = threads[id].sp
    cpus[CPU].thread = id
```