L7: Threads

Nickolai Zeldovich
6.033 Spring 2011

Recall: send with locking

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

Send and receive with yield

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

receive(bb):
    while True:
        acquire(bb.lock)
        if bb.out ≠ bb.in: …
        release(bb.lock)
        yield()
```

Yield, version 1

```
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, version 1

```
yield():
    acquire(t_lock)
    id = cpus[CPU()].thread              \
    threads[id].state = RUNNABLE          }  suspend current thread
    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, version 1

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

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

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

Yield, version 1

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

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

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

Send with yield, again

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

Send with wait / notify

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

Wait and notify

```
wait(cvar, lock):
    acquire(t_lock)
    release(lock)
    threads[id].cvar = cvar
    threads[id].state = WAITING
    yield2()        # 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
    yield2()        # 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, version 1

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

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

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

Yield version 2 (for wait)

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

switch to this CPU's kernel stack

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

choose new thread, but allow other CPUs to notify()

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

resume new thread