Computer System 09 Fall
09/16/09 Lecture 2 (2nd half)

1. Process: a running program ← typically protected

   Thread: a with a process ← share memory
   with a separate stack

\[ \begin{align*}
\text{Race conditions: } & Th1 & Th2 \\
& \{ x = x + 1 \} & \{ x = x + 1 \}
\end{align*} \]

1. TH1
   MOV X, R1
   ADD I, R1
   MOV R1, X
   MOV R1, X

1. TH2
   MOV X, R1
   ADD I, R1
   MOV R1, X
   MOV R1, X

Words fine in this order

Control section
Problem (need to protect DATA, not code)

2. To deal with race conditions: mutex — Mutual Exclusion

   POSIX Threads — pthreads
   C/C++: `pthread_mutex_t m;`
   `pthread_mutex_lock(m);`
   `pthread_mutex_unlock(m);`

   Java: `synchronized (obj)`
   `obj;`
coop: multi-threading
not much
pre-emptive MT:
disable interrupts
multi-core:
spin lock

CPU1

lock

CPU2

lock

SWAP

req

no

lock busy

SWAP

req

yes

SWAP (atomic)

0

1

Mutex

spin lock

owner queue

threads

spinlock

owner queue

 Mutex

spinlock

owner queue

CPU1

CPU2

thread 1

control block

SP

lock:

spin lock(lock)

if owner = NULL

owner = me

else

add myself to queue

sp

spinlock, release(lock)

if owner ≠ me

sleep

thread 2