Thin Locks

CS520

Dept. of Computer Science

Univ. of New Hampshire

Basic Idea

if the level of contention is expected to be low, then use lightweight locking mechanism

i.e. use cmpxchg & busy waiting

call yield if can't obtain lock after some number of tries

Intel CMPXCHG

CMPXCHG r32, m32

compare eax with m32

if equal, set ZF and store r32 in m32.

else, clear ZF and load m32 into eax.

CMPXCHG must be used with the LOCK prefix to make its two memory accesses atomic.

thinLock.s

```
#
# This provides a "thin" lock via the cmpxchg instruction.
#
# Here is the C prototype for the function:
#   int thinLock(int *lock, int tryCount);
#
#   The first parameter is the address of the memory location that is
#   serving as the lock. A zero value in the lock word means the lock
#   is available. A non-zero value means the lock is locked.
#
#   The second parameter is a count for how many attempts should be
#   made to obtain the lock before giving up and returning 0 (failure).
#
#   The function returns 1 if the lock is obtained and 0 otherwise.
#
# There is no assembly language thinUnlock because unlock is done by
# simply assigning zero to the word that is the lock.
#
      .text                     # assemble instructions
      .align  4                 # put start of function on 4-byte boundary
      .globl  thinLock          # make function name visible to linker
thinLock:
      pushl   %ebp              # save old frame pointer
      movl    %esp,%ebp         # establish new frame pointer
      pushl   %ebx              # save ebx since it is callee saved
      movl    8(%ebp), %ebx     # get first parameter into ebx
      movl    12(%ebp), %ecx    # put second parameter into ecx
tryAgain:
      movl    $0, %eax          # 0 means lock is available
      movl    $1, %edx          # put 1 into lock if it is available
      lock                      # lock the memory bus for next instruction
      cmpxchg %edx, (%ebx)      # is lock available? (ie (ebx) == eax == 0)
      je      gotLock           # if so, done
      subl    $1, %ecx          # if not, decrement counter
      je      giveUp            # if counter > 0, try again
      jmp     tryAgain
giveUp:                         # eax will be 1 if we branch here
      movl    $0, %eax          # need to return 0 however
      jmp     exit
gotLock:                        # eax will be 0 if we branch here
      movl    $1, %eax          # need to return 1 however
exit:
      popl    %ebx              # restore ebx
      popl    %ebp              # restore ebp
      ret                       # return eax
```