Concurrency #1

→ intro to problem
→ simple lock

INTRO

ATOMICITY

CONCURRENCY
Introduction: What is a thread?
- on board
  multiple "programs" executing within the SAME address space!
  usually cooperating to achieve some task, or independent related tasks
  - e.g., parallel program, web/db server
  thus, each thread has:
  - its own private set of registers
  - its own program counter
  - its own stack (and sp/bp)
  but shares
  - rest of address space (heap, static global, code section too)

What makes thread programming hard?

main-thread-0 (no locks)
  objdump -d main (inspect code)
  use this to VISUALIZE address space
  (per-thread stacks, shared parts)
  Examine in detail: main-trace.txt
  Why programs get tricky: SHARED DATA
  REAL PROBLEM: uncontrolled scheduling (interrupts at any time)
  lots of definitions:
  - program is not deterministic (indeterminate)
  - critical section
  - race condition
  - need mutual exclusion (turn indeterminate code into deterministic code)
main-thread-1 (fine-grained locks)
  need synch primitives
main-thread-2 (coarse-grained locks)
  need synch primitives but be careful
main-thread-3 (implement locks try #1: test-and-set)
  just run it
  (what is the problem?)
main-thread-4 (implement locks try #2: x86 xchg)
  how to build a lock using special hardware?
  (how to use xchg?)
main-thread-5 (implement locks try #2: x86 xchg + spinlock implementation)
  this is how
  objdump -d to look at it

Conclusions:
  Why in OS class?
  threads are basic OS primitive AND OS is a concurrent program!