Concurrency (Part II):
Mutual Exclusion, Synchronization, Deadlock, and Starvation

Professor Travis Peters
CSCI 460 Operating Systems
Fall 2019

Some slides & figures adapted from Stallings instructor resources.
Some slides adapted from Adam Bates's F'18 CS423 course @ UIUC
https://courses.engr.illinois.edu/cs423/sp2018/schedule.html

Goals for Today

Learning Objectives

• Dive a bit deeper into core topics in concurrency
• Discuss common mechanisms for achieving mutual exclusion & synchronization

Announcements

• Schedule Updates
  • Exam #1 — delayed…
  • …career fair…
  • topics to include everything up through concurrency (at least… possibly scheduling as well…)
• Exam #2 — topics will include:
  • (Scheduling)
  • Memory Management & Virtual Memory
  • File Systems & I/O
  • Selected OS Security Topics
• No Exam #3!

• Homework 2 (Chapters 3-4) DUE 10/11 — trust me, you don’t need that long!
• Homework 3 (Chapters 5-6) DUE 10/18
• Programming Assignment 1 (Concurrency + C programming/pthreads) — should be posted by Friday
Recap: Concurrency & Solutions for Mutual Exclusion

To achieve correct & meaningful solutions to concurrency problems, \textit{mutual exclusion} is a must!

Software Support
- Assume elementary mutual exclusion at the memory access level; serialized by “memory arbiter”
- Decker's Algorithm, Peterson's Algorithm

Hardware Support
- Interrupt Disabling
  - Disadvantages: inhibits processor's ability to interleave processes; doesn't work across processors.
- Special Machine Instructions
  - Compare\&Swap: compare values => if values are the same, swap!
  - Exchange (XCHG): exchanges the contents of a register w/ that of a memory location
  - Advantages: simple & easy to implement; can be used on multi-processor machines
  - Disadvantages: possibly expensive busy-waiting; starvation & deadlock are still possible

Programming Language Mechanisms
- Examples using pthreads
- Semaphores, Monitors, Condition Variables, Message Passing, Mutexes (Locks), …oh my!

\textbf{Famous Problems:} Producer/Consumer Problem, Dining Philosophers Problem

https://computing.llnl.gov/tutorials/pthreads/
Programming w/ pthreads

A good write-up on pthreads: https://computing.llnl.gov/tutorials/pthreads/

- **pthreads** (POSIX Threads)
  - defines a set of C programming language types, functions and constants.
  - is implemented with a pthread.h header and a thread library.
  - includes mutexes, condition variables, etc.
- E.g., A typical sequence in the use of a **mutex** is as follows:
  - Create and initialize a mutex variable
  - Create and start several threads
    - Several threads attempt to lock the mutex
      (only 1 succeeds and “owns” the lock)
    - The owner thread performs some set of actions
    - The owner unlocks the mutex
    - Another thread acquires the mutex and repeats the process
  - Finally, the mutex is destroyed

**NOTE:** When several threads compete for a mutex, the losers block at that call (there does exist a non-blocking call: “trylock”)

**NOTE:** It is the programmer’s responsibility to make sure every thread that needs to use a mutex does so.

For example, if 10 threads are updating the same data, but only one uses a mutex, the data can still be corrupted!
Programming w/ pthreads

A good write-up on pthreads: https://computing.llnl.gov/tutorials/pthreads/

Most of class featured in-class demo:

• <Code Walk-Through - threads01.c & threads02.c>

• What happens if I vary the number of threads?
  • 1 vs. 2 vs. 5 vs. 10 vs. ....

• What happens if the target number is small?
  • 100 vs. 1000000000000

• What happens if we don’t wait for the threads to complete?

• What happens if I compile & run on different machines (e.g., native machine/OS vs. my local Linux VM)?

• Why are these things happening?!
  • NOTE: make sure you (re-)build the executable wherever you are going to run it....
  • objdump -d t1

• With proper (simple) synchronization, why does the program appear to execute slower?