CSE 220: Systems Programming

Ethan Blanton

Department of Computer Science and Engineering
University at Buffalo
Date, Time & “Location”

The final will be **Friday, December 18, at 3:30 PM EST**.

Make-up exams are **not yet scheduled**.

The exam will be on **UBlearns**.
Format and Materials

Watch Piazza for final information on this.

The exam will be open book, open note.
(The complete list of allowed materials will be on Piazza.)

Expect:

- True/False
- Multiple choice (more than one possible answer)
- Fill-in-the-blank
- Short answer

Short answers are two words to two sentences in most cases.
Updates

Watch Piazza for updates!
The Compiler and Toolchain

- The “C compiler” is actually a chain of tools
  - We invoke the compiler driver
  - The preprocessor transforms the source code
  - The compiler turns C into assembly language
  - The assembler turns assembly language into machine code in object files
  - The linker links object files into an executable
Compiler Optimization

- Algorithmic improvements remain key.
- Knowing how the compiler works help produce better code.
- Optimizing compilers must not change semantics.
- Compilers use static information.
- We covered:
  - Constant folding
  - Code motion
  - Reduction in strength
- Procedures are problematic.
Dynamic Memory Allocation

- The OS notion of the heap is very simplistic.
- The dynamic allocator has to manage the heap.
- Metadata is required for management.
- The heap can become fragmented:
  - Internal fragmentation is inside heap blocks.
  - External fragmentation is between heap blocks.
Virtual Memory

- **Virtual memory:**
  - uses a memory management unit
  - allows the CPU to operate in a virtual address space that may be different from the physical address space
  - the MMU translates virtual addresses to physical addresses

- **Paging** is a common model for virtual memory.
  - Paged systems break both address spaces into pages.
  - Pages can be mapped individually between virtual and physical addresses.
  - Page tables allow the MMU to translate addresses.
  - Page faults bring mapped but unallocated pages into memory.
Caching and Locality

- The CPU is much faster than memory or disks.
- The difference in speeds is growing.
- Programs exhibit locality:
  - Spatial
  - Temporal
- Caching depends on locality to improve performance.
- Writing good programs requires understanding locality.
Processes, Threads, and Concurrency

- Logical control flows are execution steps through programs.
- Concurrency is multiple logical control flows at one time.
- Multiprocessing versus Multitasking
- Processes versus Threads
Races and Synchronization

- A **race** is a situation where program correctness depends on the **order of operations in concurrent flows**.
- **Data races** are races involving **modification of data**.
- **Synchronization** is the **deliberate ordering of events** in a program.
- A **critical section** is a region of code that must be accessed by at most one concurrent flow at a time.
- **Progress graphs** visualize concurrent flows.
- **Synchronization primitives**:
  - Atomic operations
  - Mutexes
  - Semaphores
  - Condition variables
- **Deadlock** is a program error caused by synchronization.
POSIX Threads and Synchronization

- The **POSIX threads** (pthreads) API provides a thread abstraction on Unix
- POSIX provides many **synchronization primitives**:
  - Mutexes
  - Semaphores
  - Condition variables
  - Thread joining
- CS:APP covers semaphores in detail
The Kernel and User Mode

- Exceptions are special control flow
- Protection domains control access to hardware resources
- Exception handlers run in supervisor mode in the kernel
- Special trap exceptions can be used to implement system calls
- System calls allow user mode programs to request access to the kernel
Input and Output

- Unix I/O is defined by the POSIX Standard
- Standard I/O is defined by the C Standard
- The kernel tracks open files with file descriptors
- All file I/O goes through the kernel
- The standard I/O library is buffered
Memory and Concurrency

- Caching and CPU architecture require more than just temporal synchronization
- Memory barriers force data visibility across cores
- Memory barriers are a hardware feature
- POSIX synchronization primitives use memory barriers
- Shared memory requires kernel assistance
Big Concepts

Tie it all together!
References I
License

Copyright 2020 Ethan Blanton, All Rights Reserved. Copyright 2019 Karthik Dantu, All Rights Reserved.

Reproduction of this material without written consent of the author is prohibited.

To retrieve a copy of this material, or related materials, see https://www.cse.buffalo.edu/~eblanton/.