CS 238P - Operating Systems

Prof. Tony Givargis

Focuses on advanced and graduate-level topics in operating systems. Presents important recent developments in operating systems, topics not covered in undergraduate operating systems courses. This includes novel operating system designs and techniques to improve existing ones.

Basics

Lecture:	TuTh 9:30-10:50 (HH 178)
Discussion:	F 12:00-12:50 (HH 178)
Office:	Tu 2:00-3:00 (DBH 3038)
Final Exam:	Thu, Dec 14, 8:00-10:00
TAs:	Mike Heddes (mheddes@uci.edu), Simon Guo (yutong4@uci.edu)
Demo Session:	Tu 2:00-4:30 (ICS1 458F)
Demo Session:	Thu/F 3:00-5:30 (ICS1 458F)
Demo Session:	F 9:30-noon (DBH 3015)
Grade:	Projects (75%), Final Exam (25%), Extra Credit (15%)
Discussion:	Canvas
Submissions:	Canvas

Rules

- Attend discussion section for details on demos and grading
- No mobile-phones during lecture (switch off or silent)
- Late submissions not accepted
- Information on this site will be updated, check often
- For academic integrity see campus policies
- For add/drop deadlines see campus policies
- Submit projects via Canvas by deadline

Lecture Topics

Here's a preliminary overview of the main themes to be discussed in our upcoming lectures. Attendance at the lectures is essential for a comprehensive understanding. Our primary objective is to delve into advanced topics in Operating Systems, aiming to enhance our precision and push the boundaries of current knowledge. It is assumed that you have [some] familiarity with the following subjects.

- Introduction
  - CPU, Caches, Memory, I/O & Storage
  - Monolitic vs Microkernel Operating Systems
  - Hard, Firm & Soft Realtime Systems
  - Bootstrap Loader (i.e., Booting)

    - Resource Management
    - Isolation & Security
    - Scheduling
- Advanced C Programming
    - Standards & Portability
    - Qualifiers, Specifiers, Declarators & Declarations
    - Expressions, Statements, Functions & Translation Units
    - Macros, Intrinsics & Inline Assembly
    - Build Essentials (i.e., Toolchain)
    - Obfuscation vs Abstraction
    - Writing Robust Code
- Process Management
    - Processes, Threads & Signals
    - Reentrant & Thread-Safe Code
    - Parallelism, Concurrency & Synchronization
    - Exception Handling & CPU Set/Jmp
    - Fork & Exec
    - Pipes & Memory Mapping
    - Linking & Loading
    - Load Balancing & CPU Affinity
- Memory Management
    - Physical vs Virtual
    - User vs Kernel
    - Heap vs Stack
    - Segfault, Process Break & Dynamic Memory Allocation
    - Non-Uniform Memory Access (NUMA)
    - Memory Barrier & Atomic Instructions
- Storage System
    - Block Devices, HDDs, SSDs & CSDs
    - I/O Scheduling, Q-Depth & Random vs Sequential I/O
    - Synchronous, Asynchronous, Direct & Scatter/Gather I/O
    - Compression, Encryption, Sharding, Replication & Erasure Codes, RAID
    - Kernel Buffers
    - File System
    - Volume Manager
- Kernel Modules & Device Drivers
    - File API & IOCTL
    - The /proc File System
    - Character vs Block Devices
    - Direct Memory Access (DMA)
    - Interrupt Service Routines (ISRs)
    - GPU/FPGA/TPU Accelerators

Projects

The five upcoming projects are quite extensive and will demand substantial research and development efforts on your part. Due to our time constraints, we will supply a substantial portion of the necessary code for each project. Nonetheless, every project will include a crucial component that you are required to develop, test, and submit to earn full credit. Given the expansive nature and complexity of these projects, we will primarily cover the introductory material and code walkthrough during lectures. Consequently, attending these lectures is highly recommended.

A compressed/archived project folder can be downloaded here: projects.zip Alternatively, try this:

- *$ wget https://ics.uci.edu/~givargis/courses/cs238p/projects.zip*

Please ensure timely submission of your projects by packaging them into a single zip file. This zip file should include the following components, all organized within a dedicated directory: (1) your source code, (2) a README.txt file containing any special comments, issues, and/or notable features, (3) if applicable, an strace output from the invocation of your program, and (4) if applicable, a valgrind report demonstrating the absence of memory leaks. Please stay tuned for further details, which will be communicated during lectures.

1. ***JIT Compiled Expression Evaluator (October 15, 2023)***

   *Design a program that accepts a mathematical expression (a string containing the symbols: +, -, *, /, %, (,), value). This program will dynamically generate a C program based on the input string, invoke a C compiler to create an equivalent loadable module, and then load and execute the machine code equivalent of the C program to produce the evaluated result of the expression. You may use the following to bootstrap the project:*

   - *Makefile*
   - *system.h*, *system.c*
   - *lexer.h*, *lexer.c*
   - *parser.h*, *parser.c*
   - *jitc.h*, *jitc.c*
   - *main.c*

   You can earn 3% extra-credit if you implement the sigmoid(double x) function that resides in your main program but is called from the generated C code to transform the final value of the expression (i.e., sigmoid(expression-value)).

2. ***Userspace Dynamic Thread Scheduler (October 29, 2023)***

   Develop a dynamic thread scheduler library with an API akin to the POSIX pthread library, providing the capability to create threads and enable cooperative concurrent execution among them. You may use the following to bootstrap the project:

   - *Makefile*
   - *system.h*, *system.c*
   - *scheduler.h*, *scheduler.c*
   - *main.c*

   You can earn 3% extra-credit if you implement an automatic context switch (e.g., every second) from one thread to another without the need for the user threads to call scheduler_yield().

3. ***Storage Class Memory Manager (November 12, 2023)***

   Create a robust memory management system featuring an API reminiscent of the C malloc() function. This system will utilize a file as its underlying storage to ensure persistent data availability across different processes. You may use the following to bootstrap the project:

   - *Makefile*
   - *system.h*, *system.c*
   - *term.h*, *term.c*

- *shell.h*, *shell.c*
- *scm.h*, *scm.c*
- *avl.h*, *avl.c*
- *main.c*

You can earn 3% extra credit by implementing the scm_free() function and showcasing the correctness of your implementation through the addition of the required code to enable a "remove" command for words.

4. ***Key/Value File System (December 3, 2023)***

   Develop a key/value file system that utilizes raw and direct I/O operations on a block device. Deploy your system on a loop-back device. Ensure that your implementation avoids reliance on kernel buffers for both reading and writing by incorporating efficient write buffering and read caching mechanisms to optimize performance. You may use the following to bootstrap the project:

   - *Makefile*
   - *system.h*, *system.c*
   - *term.h*, *term.c*
   - *device.h*, *device.c*
   - *logfs.h*, *logfs.c*
   - *kvraw.h*, *kvraw.c*
   - *index.h*, *index.c*
   - *kvdb.h*, *kvdb.c*
   - *main.c*

   You can earn 3% extra credit by implementing the ability to store and restore the state of the key/value file system on open/close. Modify the test programs to test this new behaviour.

5. ***System Performance Monitor (December 10, 2023)***

   Create a real-time system monitoring tool similar to the Unix top command that provides two different performance statistics in two different categories (e.g., CPU, memory, network, I/O). You may use the following to bootstrap the project:

   - *Makefile*
   - *system.h*, *system.c*
   - *main.c*

   You can earn 3% extra credit by adding a third category to your tool.

Tools

These tools and environments are essential for successfully completing the projects in this course. Make sure to invest the time needed to attain a proficient level of skill with these applications and systems.

- *make, gcc, gdb, gprof, gcov, nm*
- *valgrind, strace, top, iostat*
- *lsblk, dd, losetup*
- *ifconfig, wget*
- *man*
- *emacs (because everything else is boring)*
  - *.emacs*

- *Linux VM/Machine (root access)*
  - *QEMU Machine Emulator & Virtualizer*
  - *GUI Interface to QEMU*
  - *Configuration*
    - *Ubuntu Server (22.04.3 LTS)*
    - *ubuntu-22.04.3-live-server-amd64.iso*
    - *X86_64, 4 Cores, 4GB RAM, 24 GB Disk*
    - *$ sudo apt update*
    - *$ sudo apt install zip*
    - *$ sudo apt install emacs*
    - *$ sudo apt install build-essential*
    - *$ sudo apt install valgrind*
    - *$ sudo apt install wget*
    - *$ sudo apt install net-tools*

References

The internet is full of an abundance of valuable resources, encompassing e-books, write-ups, blogs, and Wiki pages that delve into the various subjects covered in this course. Below, you'll find a mini-curated selection, though it's important to note that this list is not exhaustive. You are encouraged to assemble your personal compilation of references to support your learning journey in this course and beyond.

- *Operating Systems: Three Easy Pieces*
- *The GNU C Reference Manual*
- *ANSI Escape Sequences*
- *Linux Kernel Coding Style*
- *GNU Emacs Reference Card*
- *Xv6 Home Page*

**** Commitment to an inclusive learning environment: this class adheres to the philosophy that all community members should enjoy an environment free of any form of harassment, sexual misconduct or discrimination. Please be respectful and kind to one another. ****