CS 258: Advanced Operating Systems

Sergey Bratus, Winter 2014

Objective: To confidently read code of contemporary production operating systems. To epxlore OS mechanisms that everyone uses every day, but few people know in detail, such as dynamic linking, executable and linkable format, and OS debugging support. To write working, non-trivial kernel code.

OS and HW: Illumos (fork of OpenSolaris) and Linux on x86 and x86-64.

Course directory: `http://www.cs.dartmouth.edu/~sergey/cs258/`

Grades: 50% midterm 50% final project

Lecture topics:

- Overview of UNIX system calls. The anatomy of a system call and x86 mechanisms for system call implementation. How the MMU/memory translation, segmentation, and hardware traps interact to create kernel–user context separation. What makes virtualization work.
- The kernel execution and programming context. Live debugging and tracing. Hardware and software support for debugging.
- DTrace: programming, implementation/design, internals. Kprobes and SysTrace: Linux catching up.
- Linking and loading. Executable and Linkable Format (ELF). Internals of linking and dynamic linking.
- Internals of effective spinlock implementations on x86. OpenSolaris adaptive mutexes: rationale and implementation optimization. Pre-emptive kernels. Effects of modern memory hierarchies and related optimizations.
- Process and thread kernel data structures, process table traversal, lookup, allocation and management of new structures, /proc internals, optimizations.
- Virtual File System and the layering of a file system call from API to driver. Object-orientation patterns in kernel code; a review of OO implementation generics (C++ vtables, etc).
- OpenSolaris and Linux virtual memory and address space structures. Tying top-down and bottom-up object and memory page lookups with the actual x86 page translation and segmentation. How file operations, I/O buffering, and swapping all converged to using the same mechanism.
- Kmem and Vmem allocators. OO approach to memory allocation. Challenges of multiple CPUs and memory hierarchy.
- Security: integrity, isolation, mediation, auditing. From MULTICS and MLS to modern UNIX. SELinux type enforcement: design, implementation, and pragmatics. Kernel hook systems and policies they enable. Trap systems and policies they enable. Tagged architectures and multi-level UNIX.
- ZFS overview. OpenSolaris boot environments and snapshots.
- OpenSolaris and UNIX System V system administration pragmatics: service startup, dependencies, management, system updates.
- Overview of the kernel network stack implementation. Path of a packet through a kernel. Berkeley Packet Filter architecture. Linux Netfilter architecture.

Textbook: Solaris Internals, by Jim Mauro et al (2nd ed.); Linkers and Loaders, by John Levine (`http://www.iecc.com/linker/`)