Understanding Operating Systems through Tracing and Analysis

Robert N. M. Watson
George V. Neville-Neil

November 25, 2015

1 Introduction

Operating Systems are continually evolving, complex pieces of software upon which all application software depends. While the majority of computer scientists and software developers will not, themselves, work directly on operating systems software in their careers a deep and fundamental understanding of how such systems are designed and constructed will always remain important throughout their careers. Our motivation in developing and sharing our course *Understanding Operating Systems through Tracing and Analysis* is to both broaden the number of people who understand operating systems principles, and, by extension, the principles involved in building any type of large software system. The course also acts as a base from which to continue deeper into the art of operating systems and large system design.

The course is based upon a fundamental principle, "Visibility leads to understanding." Until now most courses in operating systems have spent the majority of the students' time looking at the system from a static perspective. Learning about the basic structures that underlie operating systems by reading about them and working with either a toy or by modifying very small parts of a larger system gives some insight, but the ability to work with full blown, production quality systems is the way in which we give our students the broad, well rounded, knowledge they need to understand the systems the are interacting with and working on. Dynamic, run-time, tracing of the system is what allows our students to look at the total system, and to gain this deeper understanding.

The development of dynamic tracing [1] for use with software gives the students the ability to look at nearly any function and most structures, at run-time, allowing them to see how various application workloads interact with and be served by the operating system. Our course uses the `DTrace` dynamic tracing system with the *FreeBSD*operating system. *FreeBSD*is a modern, production quality, operating system used in many types of systems, from web and cloud computing, to storage appliances and network routers, and in small embedded devices.

Figure 1: Books

- Marshall Kirk McKusick, George V. Neville-Neil, and Robert N. M. Watson. *The Design and Implementation of the FreeBSD Operating System, 2nd Edition*, Pearson Education, Boston, MA, USA, September 2014.
- Brendan Gregg and Jim Mauro. *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD*, Prentice Hall Press, Upper Saddle River, NJ, USA, April 2011.
- Raj Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, Wiley - Interscience, New York, NY, USA, April 1991.

In this teaching guide we present the materials we use in the course, as well as guidelines for adapting the course to different teaching environments. The teaching guide and materials shared under an open source license and are continuously updated. Check our github repo for the most recent version of these documents.

2 Materials

All of the software necessary to teach the course is included in the *FreeBSD*Operating System freebsd.org. Before teaching the course on hardware it is best to start working with *FreeBSD*in a virtual environment. Instructions on how to setup *FreeBSD*with tracing in a virtual environment are given in Section 2.1.

The three books used to teach the course are listed in Figure 1. The main text about operating systems is, *The Design and Implementation of the FreeBSD Operating System, 2nd Edition.* It contains both the theory components and the explanations of how theory is turned into practice. *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD* is a guide to how to use the dynamic tracing tools with examples that can be re-purposed for the course. Finally, we include *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling* as it remains the best book we know of for explaining how to analyze large systems.

Exercises should be drawn from the main textbook, *The Design and Implementation of the FreeBSD Operating System, 2nd Edition* and may be enhanced as outlined below.

2.1 Teaching Platform

The course may be taught using either a physical platform of a virtualized one. The currently recommended hardware platform is the `BeagleBone Black`. The

`BeagleBone Black` is a small, inexpensive, single board computer built around 1Ghz, 32 bit, `ARMv7`, CPU. It has 512MB of RAM, one Ethernet port, one USB port that can be used in *target mode*, one USB peripheral port, a micro-SD slot and 4GB of on board flash. These boards are inexpensive enough, $55.00 USD as of 2015, that it is possible to provide two systems to each student for use in their lab work.

A pre-built software image can be downloaded the teachbsd site,

BeagleBone Images. To use the image copy it to an SD card using the dd command on any of FreeBSD, Linux or MacOS X. Copying an Image

The course may also be taught in a virtualized environment. Pre-built images for various virtualization systems, including `VMWare`, `qemu` and `Xen` can be downloaded from the FreeBSD project's downloads page When using a virtualized platform the sections on performance measurement should be removed as virtual machines do not provide a good platform for performance measurement.

3 Masters Course

The masters course covers both the design and implementation of the FreeBSD Operating system while also addressing current operating system research. Each of the following sub-sections has suggested readings and exercises to go along with the main course material. The goal of the course is to train graduate students in the proper experimental methods for both designing and evaluating large, complex, systems.

At the successful completion of this course students will:

- Understand operating system kernel structures.
- Design, build and evaluate experiments on a production quality kernel.
- Have exposure to the internals of processes, virtual memory, and network protocols.

4 Undergraduate Course

The goal of the undergraduate course is to give students a broad understanding of the principles of operating system design and implementation. Using an experimental approach, where students are given exercises that take them broadly across the various components of the operating system, the course emphasizes understanding large sub-systems through the use of modern software tracing tools over the creation of code for toy operating systems.

At the successful completion of this course students will understand:

- What is an operating system kernel.
- The system call interface
- The process model

- Basics of a network protocol stack
- Basic filesystem structures and how they are used

The undergraduate course is still under development.

4.1 2 Day Practitioner Course

The two day tutorial covers the use of DTrace and its application to understanding, debugging and measuring various aspects of the operating system. Student goals for the tutorial are given in Figure 2.

Figure 2: Pracitioner Teaching Goals

- Use the `dtrace` program.
- Write simple, one line, DTrace scripts
- Know the terms in the DTrace glossary (provider, probe, etc.)
- Extend scripts in the DTraceToolkit
- Work with the profile, proc, sched, and networking providers

The two day tutorial covers the use of DTrace and delves into several of the kernel subsystems. The outline of the two day tutorial is shown in Figure 3

Each section contains a set of lab exercises. Each lab is intended to last 30 minutes, with 20 minutes of work time and 10 minutes of followup discussion. It is important to solicit scripts and one liners from the students to demonstrate to the rest of the class as well as using scripts and one liners of your own.

Figure 3: Two Day Course Outline

Introduction Goals, tracing over view, and history

DTrace Command, Glossary, One Liners

Processes Process Model, Process Lifecycle, fork, exec, exit, signals

Scheduler Process States, Sched Provider, Running Threads

Locking Lock Types, Lock Provider, Lock Stat Collection

Networking Sockets, UDP, TCP, TCP States, Packet Forwarding

Filesystem Naming, Name Cache, VNODEs, VFS

4.2 5 Day Practitioner Course

The five day tutorial includes all of the material in two day version, Section 3, as well as extra modules covering more of the storage and event subsystems. The complete outline for the five day tutorial is shown in Figure 4.

Figure 4: Five Day Tutorial Outline

Introduction Course goals, tracing over view, and history

DTrace Command, Glossary, One Liners

Processes Process Model, Process Lifecycle, fork, exec, exit, signals

Scheduler Process States, Sched Provider, Running Threads

Locking Lock Types, Lock Provider, Lock Stat Collection

Networking Sockets, UDP, TCP, TCP States, Packet Forwarding

Filesystem Naming, Name Cache, VNODEs, VFS

GEOM Overview, Layering, Tracking Requests, Adding GELI

Events Select, Poll and KQUEUE

The five day course is still under development.

5 Further Resources

TeachBSD Project	|https://www.teachbsd.org/—
FreeBSD Project	`https://www.FreeBSD.org/`
FreeBSD Source Code Repository	`http://svn.FreeBSD.org/`
DTrace on FreeBSD	`https://wiki.freebsd.org/DTrace`
FreeBSD and Linux Kernel Cross-Reference	`http://fxr.watson.org/`
FreeBSD Benchmark Advice	`https://wiki.freebsd.org/BenchmarkAdvice`
BeagleBone Black	`http://beagleboard.org/black`

References

[1] Bryan Cantril, Michael Shapiro, and Adam Leventhal. Dynamic Instrumentation of Production Systems. 2004.