SEE TEXT ONLY

TEACHING Operating Systems with FreeBSD Through Tracing, Analysis, & Experimentation

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

Many people who study computer science at universities encounter their first truly large system when studying operating systems. Until their first OS course, their projects are small, self-contained, and often written by only one person or a team of three or four. Since the first courses on operating systems were begun back in the 1970s, there have been three ways in which such classes have been taught. At the undergraduate level, there is the "trial by fire," in which students extend or recreate classical elements and forms of OS design, including kernels, processes, and filesystems. In trial-by-fire courses the students are given a very large system to work with and they are expected to make small, but measurable, changes to it. Handing someone a couple million lines of C and expecting them to get something out of changing a hundred lines of it seems counterintuitive at the least. The second undergraduate style is the "toy system." With a toy system the millions of lines are reduced to some tens of thousands, which makes understanding the system as a whole easier, but severely constrains the types of problems that can be presented, and the lack of fidelity, as compared to a real, fielded operating system, often means that students do not learn a great deal about operating systems, or large systems in general. For graduate students, studying operating systems is done through a research readings course, where students read, present, discuss, and write about classic research where they are evaluated on a term project and one or more exams.

Fig 1. BeagleBone Black board used in teaching. The single USB cable is used to provide both power and communications with a lab workstation or student notebook computer.

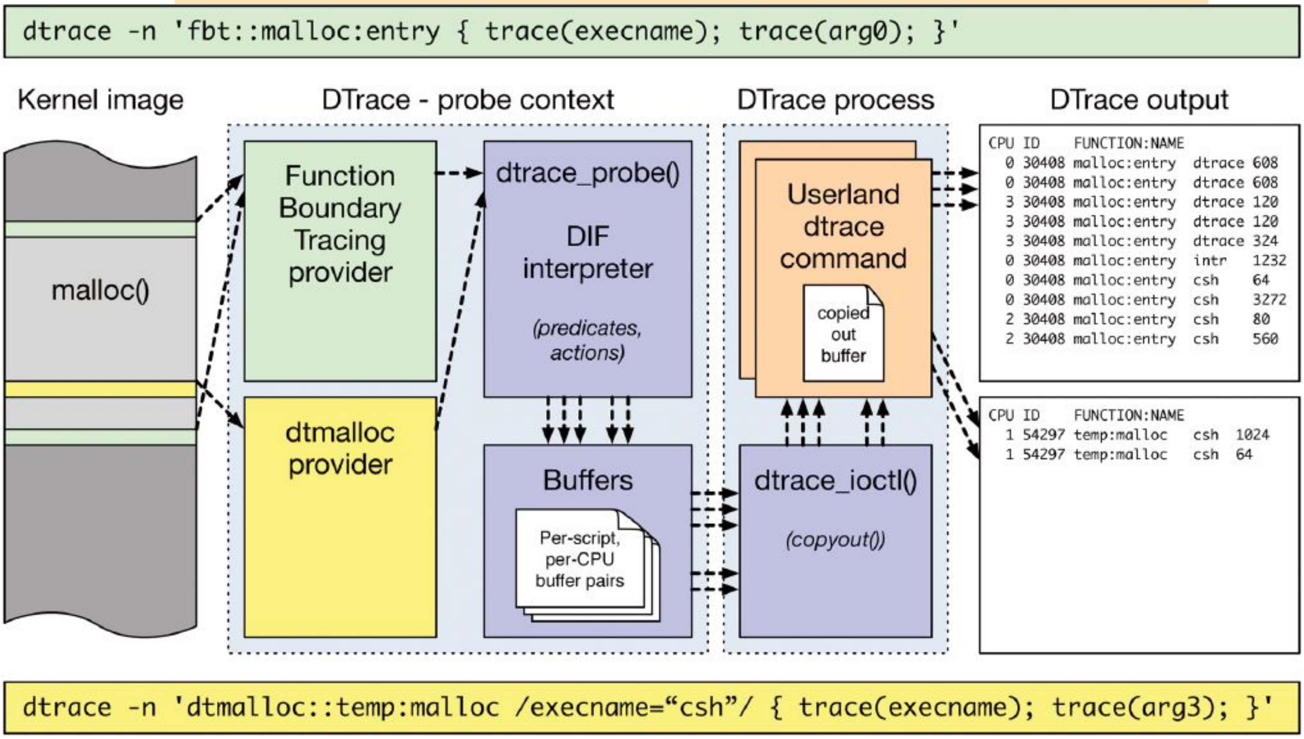
For practitioners, those who have already left the university, or those who entered computer science from other fields, there have been even fewer options. One of the few examples of a course aimed at practicing software engineers is the series "FreeBSD Kernel Internals" by Marshall Kirk McKusick, with whom both authors of this article worked on the most recent edition of *The Design and Implementation of the FreeBSD Operating System*. In the "FreeBSD Kernel Internals" courses, students are walked through the internals of the FreeBSD operating system with a generous amount of code reading and review, but without modifying the system as part of the course.

For university courses at both the undergraduate and graduate level, we felt there had to be a middle way where we could use a real-world artifact such as FreeBSD, which is deployed in products around the world, while making sure the students didn't get lost in the millions of lines of code at their disposal.

Deep-dive Experimentation

Starting in 2014, Robert and George undertook to build a "deep-dive experimentation" course for graduate students taught by Robert N. M. Watson at the University of Cambridge, as well as a practitioner course taught at conferences in industrial settings by George Neville-Neil.

In the deep-dive course, students learn about and analyze specific CPU/OS/protocol behaviors using tracing via DTrace and performance using the hwpmc(4) system. Using tracing to teach mitigates the risk of OS kernel hacking in a short course, while allowing the students to work on real-world systems rather than toys. For graduate students, we target research skills and not just OS design. The deep-dive



```
dtrace -n 'fbt::malloc:entry { trace(execname); trace(arg0); }'
```

```
CPU ID     FUNCTION:NAME
  0 30408 malloc:entry  dtrace 608
  0 30408 malloc:entry  dtrace 608
  3 30408 malloc:entry  dtrace 120
  3 30408 malloc:entry  dtrace 120
  3 30408 malloc:entry  dtrace 324
  0 30408 malloc:entry  intr   1232
  0 30408 malloc:entry  csh    64
  0 30408 malloc:entry  csh    3272
  2 30408 malloc:entry  csh    80
  2 30408 malloc:entry  csh    560
```

```
CPU ID     FUNCTION:NAME
  1 54297 temp:malloc   csh  1024
  1 54297 temp:malloc   csh  64
```

```
dtrace -n 'dtmalloc::temp:malloc /execname="csh"/ { trace(execname); trace(arg3); }'
```

Fig. 2. DTrace is a critical part of the course's teaching approach—students trace kernels and applications to understand their performance behavior. They also need to understand—at a high level—how DTrace works, in order to reason about the "probe effect" on their measurements.

course is only possible due to development of integrated tracing and profiling tools, including DTrace and CPU performance counters present in FreeBSD.

The aims of the graduate course include teaching the methodology, skills, and knowledge required to understand and perform research on contemporary operating systems by teaching systems-analysis methodology and practice, exploring real-world systems artifacts, developing scientific writing skills, and reading selected original systems research papers.

The course is structured into a series of modules. Cambridge teaches using 8-week academic terms, providing limited teaching time compared to US-style 12-to-14-week semesters. However, students are expected to do substantial work outside of the classroom, whether in the form of reading, writing, or lab work. For the Cambridge course, we had six one-hour lectures in which we covered theory, methodology, architecture, and practice, as well as five two-hour labs. The labs included 30 minutes of extra teaching time in the form of short lectures on artifacts, tools, and practical skills. The rest of the students' time was spent doing hands-on measurement and experimentation. Readings were also assigned, as is common in graduate level courses, and these included both selected portions of module texts and historic and contemporary research papers. Students produced a series of lab reports based on experiments done in (and out) of labs. The lab reports are meant to refine scientific writing style to make it suitable for systems research. One practice run was marked, with detailed feedback given, but not assessed, while the following two reports were assessed and made up 50% of the final mark.

Three textbooks are used in the course, including *The Design and Implementation of the FreeBSD Operating System, 2nd Edition*, as the core operating systems textbook; *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, which shows the students how to measure and evaluate their lab work; and *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD*, covering the use of the DTrace system.

Although many courses are now taught on virtual machine technology, we felt it was important to give the students experience with performance measurement. Instead of equipping a large room of servers, we decided, instead, to teach with one of the new and inexpensive embedded boards based around the ARM series of processors. Initially we hoped to use the Raspberry Pi as it is popular, cheap, and designed at the same university at which the course would first be taught. Unfortunately, the RPi available at the time did not have proper performance counter support in hardware due to a feature being left off the system-on-chip design when it was originally produced. With the RPi out of the running, we chose the BeagleBone Black, which is built around a 1-GHz, 32-bit ARM Cortex A-8, a superscalar processor with MMU and L1/L2 caches. Each student had one of these boards on which to do lab work. The BBB has serial console and network via USB. We provided the software images on SD cards that formed the base of the students' lab work. The software images contain the FreeBSD operating system, with DTrace and support for the on-board CPU performance counters, and a set of custom micro-benchmarks. The benchmarks are used in the labs and cover areas such as POSIX I/O, POSIX IPC, and networking over TCP.

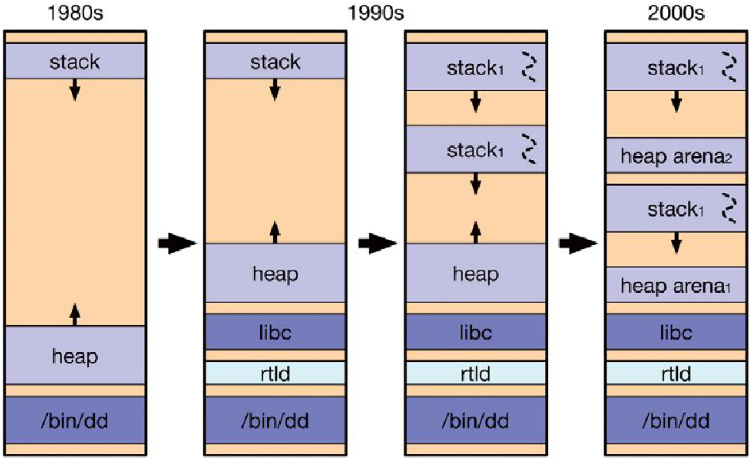


Fig 3. Students learn not just about the abstract notion of a UNIX "process," but also the evolution of the approach over the decades: dynamic linking, multithreading, and contemporary memory allocators such as FreeBSD's "jemalloc."

Eight Weeks, Three Sections

The eight weeks of the course are broken up into three major sections. In weeks one and two, there is a broad introduction to OS kernels and tracing. We want to give the students a feel for the system they are working on and the tools they'll be working with. During these first two weeks, students are assigned their first lab, in which they are expected to look at POSIX I/O performance. I/O performance is measured using a synthetic benchmark we provide in which they look at file block I/O using a constant total size with a variable buffer size. The conventional view is that increasing the buffer size will result in fewer system calls and improved overall performance, but that is not what the students will find. As buffer sizes grow, the working set first overflows the last-level cache, preventing further performance growth, and later exceeds the superpage size, measurably decreasing performance as page faults require additional memory zeroing.

The second section, covering weeks three through five, is dedicated to the process model. As the process model forms the basis of almost all modern programming systems, it is a core component of what we want the students to be able to understand and investigate during the course and afterwards in their own research. While learning about the process model, the students are also exposed to their first micro-architectural measurement lab in which they show the implications of IPC on L1 and L2 caching. The micro-architectural lab is the first one that contributes to their final grade.

The last section of the course is given over to networking, specifically the Transport Control Protocol (TCP). During weeks six through eight, the students are exposed to the TCP state machine and also measure the effects of latency on bandwidth in data transfers.

Challenges and Refinements

The graduate course has been taught twice at Cambridge, and we have reached out to other universities to talk with them about adopting the material we have produced. In teaching the course, we discovered many things that worked, as well as a few challenges to be overcome as the material is refined. We can

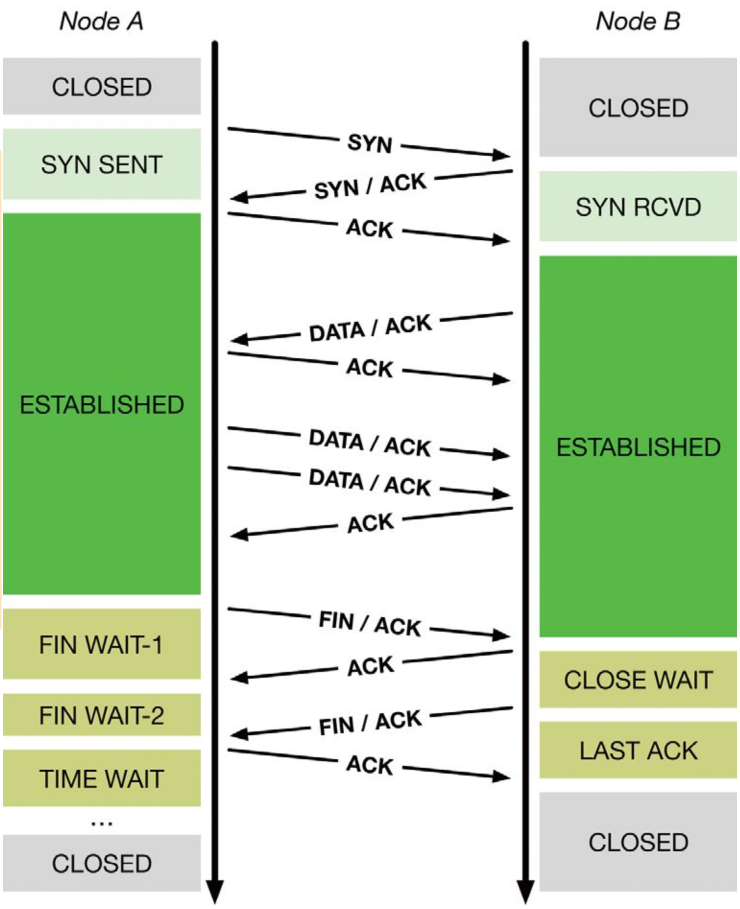


Fig 4. Labs 3 and 4 of the course require students to track the TCP state machine and congestion control using DTrace, and to simulate the effects of latency on TCP behavior using FreeBSD's DUMMYNET traffic control facility.

confirm that tracing is a great way to teach complex systems because we were able to get comprehensive and solid lab reports/analysis from the students, which was the overall goal of the course. The students were able to use cache hit vs. system-call rates to explain IPC performance. They produced TCP time-sequence plots and graphical versions of the TCP state machine all from trace output. Their lab reports had real explanations of interesting artifacts, including probe effects, superpages, DUMMYNET timer effects, and even bugs in DTrace. Our experiment with using an embedded board platform worked quite well—we could not have done most of these experiments on VMs. Overall, we found that the labs were at the right level of difficulty, but that too many experimental questions led to less focused reports—a concern addressed in the second round of teaching.

On the technical side, we should have committed to one of R, Python, or iPython Notebooks for use by the students in doing their experimental evaluations and write-ups. Having a plethora meant that there were small problems in each, all of which had to be solved and which slowed down the students' progress. When teaching the course for the first time, there were several platform bumps, including USB target issues, DTrace for ARMv7 bugs, and the 4-argument limitation for DTrace on ARMv7.

Teaching Practitioners

Teaching practitioners differs from teaching university students in several ways. First, we can assume more background, including some knowledge of programming and experience with Unix. Second, practitioners often have real problems to solve, which can lead these students to be more focused and more involved in the course work. We can't assume everything, of course, as most of the students will not have been exposed to Kernel Internals or have a deep understanding of corner cases.

Our goals for the practitioner course are to familiarize people with the tools they will use, including DTrace, and to give them practical techniques for dealing with their problems. Along the way we'll educate them about how the OS works and dispel their fears of ever understanding it. Contrary to popular belief, education is meant to dispel the students fear of a topic so that they can appreciate it more fully and learn it more deeply.

The practitioner's course is currently two eight-hour days. The platform is the student's laptop or a virtual machine. First taught at AsiaBSDCon 2015 and AsiaBSDCon 2016, the next course will be at BSDCan 2016.

Five-day, 40-hour course Hardware or VM Platform Video Recordings

Like the graduate-level course, this course is broken down into several sections and follows roughly the same narrative arc. We start by introducing DTrace using several simple and yet powerful "one liners." A DTrace one liner is a single command that yields an interesting result.

Figure below shows an example one-liner wherein every name lookup on the system is shown at run time.

```
dtrace -n 'vfs:namei:lookup:entry \
               { printf("%s", stringof(arg1));}'
      CPU      ID FUNCTION:NAME
        2   27847 lookup:entry /bin/ls
        2   27847 lookup:entry /libexec/ld-elf.so.1
        2   27847 lookup:entry /etc
        2   27847 lookup:entry /etc/libmap.conf
        2   27847 lookup:entry /etc/libmap.conf
```

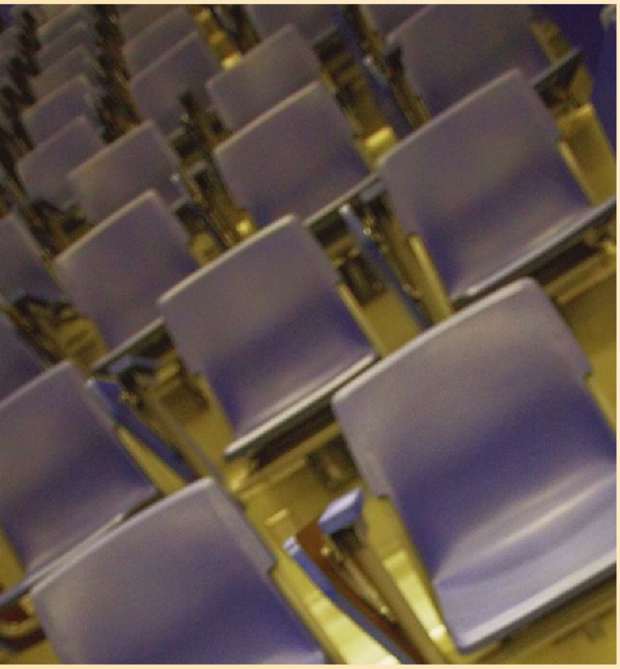
The major modules are similar to the university course and cover locking, scheduler, files and the filesystem, and finally networking. The material is broken up so that each one-hour lecture is followed by a 30 minute lab in which students use the VMs on their laptops to modify examples given during the lectures or solve a directed problem. Unlike classes where we have access to hardware, the students do not take any performance measurements with hwpmc(4) since the results would be unreliable and uninformative.

Having taught the practitioner course several times, we have learned a few things. Perhaps the most surprising was that the class really engages the students. Walking around the class during the labs, we didn't see a single person checking email or reading social media—they were actually solving the problems. The students often came up with novel answers to the problems presented, and this was only after being exposed to DTrace for a few hours. Their solutions were interesting enough that we integrated them back into the teaching during the next section. Finally, and obvious from the outset, handing a pre-built VM to the students significantly improves class startup time, with everyone focused on the task at hand, rather than tweaking their environment. Since the FreeBSD Project produces VM images for all the popular VM systems along with each release, it is easy to have the students pre-load the VM before class, or to hand them one on a USB stick when they arrive.

It's All Online!

With the overall success of these courses, we have decided to put all the material online using a permissive, BSD-like publishing license. The main page, can be found at www.teachbsd.org and our github repo, which contains all our teaching materials for both the graduate and practitioner courses can be found in github: https://github.com/teachbsd/course, where you can fork the material for your own purposes as well as send us pull requests for new features or any bugs found in the content. We would value your feedback on, and suggestions for improvements to, the course—and please let us know if you are teaching with it! ●

GEORGE V. NEVILLE-NEIL works on networking and operating system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are code spelunking, operating systems, networking and time protocols. He is the coauthor with Marshall Kirk McKusick and Robert N. M. Watson of *The Design and Implementation of the FreeBSD Operating System*. For over 10 years he has been the columnist better known as Kode Vicious. He earned his bachelor's degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. He is an avid bicyclist and traveler and currently lives in New York City.



DR. ROBERT N. M. WATSON is a University Lecturer in Systems, Security, and Architecture at the University of Cambridge Computer Laboratory; FreeBSD developer and core team member; and member of the FreeBSD Foundation board of directors. He leads a number of cross-layer research projects spanning computer architecture, compilers, program analysis, program transformation, operating systems, networking, and security. Recent work includes the Capsicum security model, MAC Framework used for sandboxing in systems such as Junos and Apple iOS, and multithreading in the FreeBSD network stack. He is a coauthor of *The Design and Implementation of the FreeBSD Operating Systems (Second Edition)*.