Lab 7: Final JOS project

Handed out Wednesday, Oct 22, 2014
Piazza Discussion Due, Oct 31, 2014.
Proposals Due, Nov 7, 2014.
Code repository Due, Dec 5, 2014
Check-off and in-class demos, Week of Dec 10, 2014

Introduction

For the final project you have two options:

- Work on your own and do lab 6, including one challenge exercise in lab 6. (You are free, of course, to extend lab 6, or any part of JOS, further in interesting ways, but it isn't required.)
- Work on in a team of one, two or three, on a project of your choice that involves your JOS. This project must be of the same scope as lab 6 or larger (if you are working in a team)

The goal is to have fun and explore more advanced O/S topics; you don't have to do novel research.

If you are doing your own project, we'll grade you on how much you got working, how elegant your design is, how well you can explain it, and how interesting and creative your solution is. We do realize that time is limited, so we don't expect you to re-write Linux by the end of the semester. Try to make sure your goals are reasonable; perhaps set a minimum goal that's definitely achievable (e.g., something of the scale of lab 6) and a more ambitious goal if things go well.

If you are doing lab 6, we will grade you on whether you past the tests and the challenge exercise.

Deliverables

Oct 31: Piazza discussion and form groups of 1, 2, or 3 (depending on which final project option you are choosing). Use the lab7 tag/folder on Piazza. Discuss ideas with others in comments on their Piazza posting. Use these postings to help find other students interested in similar ideas for forming a group. Course staff will provide feedback on project ideas on Piazza; if you'd like more detailed feedback, come chat with us in person. (If you do lab 6, just let us know you are doing lab 6.)

Nov 7: Submit a proposal at the course website, just a paragraph or two. The proposal should include your group members list, the problem you want to address, how you plan to address it, and what are you proposing to specifically design and implement. (If you are doing lab 6, there is nothing to do for this deliverable.)

Dec 5: submit source code along with a brief write-up. Put the write-up under the top-level source directory with the name "`README.pdf`". Since some of you will be working in groups for this lab assignment, you may want to use git to share your project code between group members. You will need to decide on whose source code you will use as a starting point for your group project. Make sure create a branch for your final project, and name it **lab7**. (If you do lab 6, follow the lab 6 submission instructions.)

Week of Dec 10: short in-class demonstration. Prepare a short in-class demo of your JOS project. We will provide a projector that you can use to demonstrate your project. Depending on the number of groups and the kinds of projects that each group chooses, we may decide to limit the total number of presentations, and some groups might end up not presenting in class.

Week of Dec 8: check-off with TAs. Demo your project to the TAs so that we can ask you some questions and find out in more detail what you did.

Project ideas

If you are not doing lab 6, here's a list of ideas to get you started thinking. But, you should feel free to pursue your own ideas. Some of the ideas are starting points and by themselves not of the scope of lab 6, and others are like to be much of larger scope.

- Build a virtual machine monitor that can run multiple guests (for example, multiple instances of JOS), using x86 VM support.
- Do something useful with the x86 Trusted Execution Technology. For example, run applications without having to trust the kernel. Here is a recent paper on this topic.
- Do something useful with the hardware protection of Intel SGX. Here is a recent paper using Intel SGX.
- Make the JOS file system support writing, file creation, logging for durability, etc., perhaps taking ideas from Linux EXT3.
- Use file system ideas from Soft updates, WAFL, ZFS, or another advanced file system.
- Add snapshots to a file system, so that a user can look at the file system as it appeared at various points in the past. You'll probably want to use some kind of copy-on-write for disk storage to keep space consumption down.
- Build a distributed shared memory (DSM) system, so that you can run multi-threaded shared memory parallel programs on a cluster of machines, using paging to give the appearance of real shared memory. When a thread tries to access a page that's on another machine, the page fault will give the DSM system a chance to fetch the page over the network from whatever machine currently stores.
- Allow processes to migrate from one machine to another over the network. You'll need to do something about the various pieces of a process's state, but since much state in JOS is in user-space it may be easier than process migration on Linux.
- Implement paging to disk in JOS, so that processes can be bigger than RAM. Extend your pager with swapping.
- Implement mmap() of files for JOS.
- Use xfi to sandbox code within a process.
- Support x86 2MB or 4MB pages.
- Modify JOS to have kernel-supported threads inside processes. See in-class uthread assignment to get started. Implementing scheduler activations would be one way to do this project.
- Use fine-grained locking or lock-free concurrency in JOS in the kernel or in the file server (after making it multithreaded). The linux kernel uses read copy update to be able to perform read operations without holding locks. Explore RCU by implementing it in JOS use it to support a name cache with lock-free reads.
- Implement ideas from the Exokernel papers, for example the packet filter.
- Make JOS have soft real-time behavior. You will have to identify some application for which this is useful.
- Make JOS run on 64-bit CPUs. This includes redoing the virtual memory system to use 4-level pages tables. See reference page for some documentation.
- Port JOS to a different microprocessor. The osdev wiki may be helpful.
- A window system for JOS, including graphics driver and mouse. See reference page for some documentation. sqrt(x) is an example JOS window system (and writeup).
- Implement dune to export privileged hardware instructions to user-space applications in JOS.
- Intel recently announced transactional memory support for its upcoming processors. Implement support for Intel's TSX in the QEMU emulator. A follow-on project would be to explore the use of Intel TSX primitives in writing concurrent software, such as extending the JOS kernel to use transactional memory.

- Write a user-level debugger; add strace-like functionality; hardware register profiling (e.g. Oprofile); call-traces
- Binary emulation for (static) Linux executables