CSE 506 Research Project Ideas

These are some research-oriented project ideas which may involve some measure of collaboration with students in OSCAR or another lab. Please do not distribute this list outside of Stony Brook, and do not contact students directly without talking to me first.

Graphene LibOS and content addressable storage (CAS)

Graphene is a Linux library OS---most of the kernel functionality is actually implemented as a library between libc and a very small system call table. We are developing Graphene in the OSCAR lab, and it needs lots of features. We are also exploring the use of content-addressable storage in Graphene.

- The paper Dune: Safe User-level Access to Privileged CPU Features allows a process to directly manipulate page tables, interrupts, etc. This would be a useful feature for the Graphene Library OS. This project would involve getting Dune to work on a test system and adding support to Graphene to recieve hardware exceptions directly.

- Implement several optimizations for the back-end of our content-addressible storage service. This would involve exploring several disk layout options for the low-level storage of data.
- Convert git to use our content-addressable storage system as its backing store for objects.
- Port our storage server, or an X server to run on Graphene. What additional host interfaces are needed for hardware accesses?
- Port firefox, or another interesting application to Graphene.

VM Introspection

VM Introspection is where an agent outside of a virtual machine (VM) attempts to infer what is going on inside the VM. A common approach to this is to read guest physical memory and overlay it with guest OS data structures. As you might guess, this is error prone.

This project would involve trying to use nested paging to unmap portions of the guest physical memory that the currently running process is probably not using, then using page faults to learn which regions of physical memory are being touched by the process and its system calls. This project would involve substantial KVM page table hacking.

Integrate swapping heuristics with the disk and/or CPU scheduler.

As we saw in class, swapping heuristics in Linux generally do not factor in any information about what other schedulers are doing. Explore how things like the runqueue or disk queues could be leveraged to make smarter decisions about what to swap. For instance, pdflush might take the least-recently-used page from the last process in the run queue, or might attempt to retract read-ahead requests that are in the disk queue. Do any of these heuristics do a better job of ensuring applications make progress under system memory pressure?

Verification

If you happen to be an expert at verification or formal methods, I have a specific algorithm I'd be quite interested in proving correct.

Cure cancer, or something

At OSDI 12, the keynote speaker was Jeff Haussler of the UCSC cancer genomics hub. He expressed a need for collaboration between systems researchers and researchers in the biological sciences.

According to Dr. Haussler, "There are two types of problems. One is I give one person's personal genome, and you're supposed to interpret it. The other is that you have a database of thousands or millions of genomes and you're supposed to search that for patterns that are common and associated with certain kinds of disease attributes or clinical response. Both of them are incredibly important types of problems that are distinct."

The core task of the second problem is to preprocess a large stream of text in order to efficiently query it for matches. This sounds a lot like data deduplication to me.

Inline data deduplication is a form of compression where data is broken into segments (called chunks), which are indexed by their contents (usually a SHA-1 hash). When new data is received, the new chunks are compared against the database of existing chunks in order to find content matches. When a match occurs, there is no reason to write the data a second time --- just use a pointer to the existing -- --block.

The question here is how do we break the data into chunks? The data is a long sequence of the letters (A, C, G, and U), and each sequence contains many different types of mutations. A single base can be flipped (a point mutation). A section of DNA can be deleted, duplicated, or rearranged. A new section can be inserted where there previously was none. Depending on the rate of mutations, our choice of chunk sizes will impact our ability to detect matches. If we have chunks that are too large, no 2 chunks will match. But if we have chunks that are too small, then we might as well compare the strings a base at a time...

What this project will explore is content-defined chunking. There are already existing algorithms to do this, whose input parameters determine average chunk size. You will be applying these algorithms, selecting the most appropriate ones and adapting them to compete in the CGHub bakeoff. This is a global competition. The data is availbale on UCSC CGHub for you to explore. You will be required to submit to the competition, and present your scheme, parameters, and code.

Resources:

- OSDI keynote: https://www.usenix.org/conference/osdi12/tech-schedule/osdi-12-program (the video is posted for you to stream or download)
- A Low-bandwidth Network File system (introduces content defined chunking in a digestible way, and **whoah! source code**) Muthitacharoen, Chen, Mazieres
- A framework for analyzing and improving content-based chunking algorithms (a good read, introduces Two-thresholds two-divisors, a minor improvement to what LBFS does) Eshghi, Tang
- Identifying almost identical files using context triggered piecwise hashing (finds homologous files, not required to be exactly identical) Kornblum