Multiprocessor Weenix

Eric Caruso, Jackson Owens

We attempted to understand what it takes to port a single-core operating system so that it might take advantage of all available cores in a computer. Using cs169's Weenix operating system as a starting point, we successfully got the computer to boot and start running programs in user-land across multiple cores. We felt that this is a good step towards learning what more modern operating system designs have to cope with, as Weenix is based on older operating systems and might become less relevant as systems change.

Weenix is built on the assumptions that it runs on one core and is fully cooperative, so there are often many liberties taken in the code that make it difficult to translate into something SMP-safe. In order to remedy this, we created synchronization primitives and data structures that used locks, and had to learn proper use of memory barriers. We also tried to see if we could use lock-free structures to keep contention down, and the implementation of lock-free data structures is complicated by the absence of a garbage collector in this instance. We also implemented a better scheduler which took account of thread priority and interactivity, which was based on Linux's O(1) scheduler, and dealt with the problem of work stealing.

This work might be used as a teaching tool to see some of the pitfalls and difficulties with designing multiprocessor operating systems.