Lessons from Minix

COSC 310 Operating Systems | Soobin Rho

General Thoughts

I was thinking about the fact that the lawyers at Bell Lab did not like Unix V6’s source code being studied by university students. They even decided to make Unix V7 close sourced. If I were them, I would have acted in the complete opposite way.

However, here’s a what if. What if the lawyers at Bell Lab did not oppose to it being studied widely by university students? What if they liked it and embraced it? In such an alternative universe, it seems to me that Unix V7 and later versions would have been much more successful than they actually were. Minix and Linux, in turn, would not have been as successful. So, maybe this means the lawyers at Bell Lab made the right choice for everyone after all.

Micro vs Monolithic Kernel Debate

I strongly believe we need to keep both the micro kernel and the monolithic kernel. Both have advantages and disadvantages such that it’s extremely useful to have the option to choose either of them depending on the need.

In fact, the following is a summary of what I understood from the article:

The micro kernel’s biggest strength is reliability. “[In monolithic kernels,] a single bug in any component could crash the whole system … In the world of embedded computing, where reliability often trumps performance, microkernels dominate.” (pg. 75) "In a monolithic system like Windows or Linux, a rogue or malfunctioning audio driver has the power to erase the disk; in MINIX, the microkernel will not let it." (pg. 76)

In contrast, the monolithic kernel’s biggest strength is speed. “On January 29, 1992, I [Andrew Tanenbaum] posted a message to comp.os.minix saying microkernels were better than monolithic designs, except for performance.” (pg. 74)

So, what I consider to be a good idea is to use the micro kernel when we need that reliability, but use the monolithic kernel if we were to prioritize speed.

By the way, one extra advantage the micro kernel has that I really like is the ease of development in terms of future portability: “[I, Andrew Tanenbaum,] try to make the system port easily to future hardware … [For example,] I could have taken shortcuts and put everything into one executable but decided against it because I wanted the design to work on future CPUs with an MMU.” (pg. 72) Also, it helps that “the microkernel was indeed small. Only the scheduler, low-level process management, interprocess communication, and the device drivers were in it,” (pg. 72) which means it tends to be easier to debug and engineer from the standpoint of a developer, compared to the monolithic kernel.

How My Study of Operating Systems Concepts Changed How I Understand This Article

I immediately felt a difference when I was reading the article: the context. I was so much better able to understand the concepts discussed in the article because I already had the context – e.g. scheduling algorithms and racing conditions, which we learned in class.

This reinforces my old-time idea that the more you know, the easier it is to learn more. This, in turn, made me contemplate on my future career options. Yes, working right away as a software engineer or whatever I aspire to be in the industry will be nice. However, that would mean I would lose the opportunity cost to have more knowledge – i.e. more context for whichever field I can pursue in a grad school. It is a hard question.

Anyways, I am glad we read this article in class. It was a fun read. I think it was largely thanks to how Andrew Tanenbaum writes. His writings are personal and relatable, so was super enjoyable to read.