11

Summary Dialogue on CPU Virtualization

Professor: *So, Student, did you learn anything?*

Student: *Well, Professor, that seems like a loaded question. I think you only want me to say "yes."*

Professor: *That's true. But it's also still an honest question. Come on, give a professor a break, will you?*

Student: *OK, OK. I think I did learn a few things. First, I learned a little about how the OS virtualizes the CPU. There are a bunch of important* ***mechanisms*** *that I had to understand to make sense of this: traps and trap handlers, timer interrupts, and how the OS and the hardware have to carefully save and restore state when switching between processes.*

Professor: *Good, good!*

Student: *All those interactions do seem a little complicated though; how can I learn more?*

Professor: *Well, that's a good question. I think there is no substitute for doing; just reading about these things doesn't quite give you the proper sense. Do the class projects and I bet by the end it will all kind of make sense.*

Student: *Sounds good. What else can I tell you?*

Professor: *Well, did you get some sense of the philosophy of the OS in your quest to understand its basic machinery?*

Student: *Hmm... I think so. It seems like the OS is fairly paranoid. It wants to make sure it stays in charge of the machine. While it wants a program to run as efficiently as possible (and hence the whole reasoning behind* ***limited direct execution****), the OS also wants to be able to say "Ah! Not so fast my friend" in case of an errant or malicious process. Paranoia rules the day, and certainly keeps the OS in charge of the machine. Perhaps that is why we think of the OS as a resource manager.*

Professor: *Yes indeed — sounds like you are starting to put it together! Nice.*

Student: *Thanks.*

Professor: *And what about the policies on top of those mechanisms — any interesting lessons there?*

Student: *Some lessons to be learned there for sure. Perhaps a little obvious, but obvious can be good. Like the notion of bumping short jobs to the front of the queue — I knew that was a good idea ever since the one time I was buying some gum at the store, and the guy in front of me had a credit card that wouldn't work. He was no short job, let me tell you.*

Professor: *That sounds oddly rude to that poor fellow. What else?*

Student: *Well, that you can build a smart scheduler that tries to be like SJF and RR all at once — that MLFQ was pretty neat. Building up a real scheduler seems difficult.*

Professor: *Indeed it is. That's why there is still controversy to this day over which scheduler to use; see the Linux battles between CFS, BFS, and the O(1) scheduler, for example. And no, I will not spell out the full name of BFS.*

Student: *And I won't ask you to! These policy battles seem like they could rage forever; is there really a right answer?*

Professor: *Probably not. After all, even our own metrics are at odds: if your scheduler is good at turnaround time, it's bad at response time, and vice versa. As Lampson said, perhaps the goal isn't to find the best solution, but rather to avoid disaster.*

Student: *That's a little depressing.*

Professor: *Good engineering can be that way. And it can also be uplifting! It's just your perspective on it, really. I personally think being pragmatic is a good thing, and pragmatists realize that not all problems have clean and easy solutions. Anything else that caught your fancy?*

Student: *I really liked the notion of gaming the scheduler; it seems like that might be something to look into when I'm next running a job on Amazon's EC2 service. Maybe I can steal some cycles from some other unsuspecting (and more importantly, OS-ignorant) customer!*

Professor: *It looks like I might have created a monster! Professor Frankenstein is not what I'd like to be called, you know.*

Student: *But isn't that the idea? To get us excited about something, so much so that we look into it on our own? Lighting fires and all that?*

Professor: *I guess so. But I didn't think it would work!*