12

A Dialogue on Memory Virtualization

Student: *So, are we done with virtualization?*

Professor: *No!*

Student: *Hey, no reason to get so excited; I was just asking a question. Students are supposed to do that, right?*

Professor: *Well, professors do always say that, but really they mean this: ask questions, **if** they are good questions, **and** you have actually put a little thought into them.*

Student: *Well, that sure takes the wind out of my sails.*

Professor: *Mission accomplished. In any case, we are not nearly done with virtualization! Rather, you have just seen how to virtualize the CPU, but really there is a big monster waiting in the closet: memory. Virtualizing memory is complicated and requires us to understand many more intricate details about how the hardware and OS interact.*

Student: *That sounds cool. Why is it so hard?*

Professor: *Well, there are a lot of details, and you have to keep them straight in your head to really develop a mental model of what is going on. We'll start simple, with very basic techniques like base/bounds, and slowly add complexity to tackle new challenges, including fun topics like TLBs and multi-level page tables. Eventually, we'll be able to describe the workings of a fully-functional modern virtual memory manager.*

Student: *Neat! Any tips for the poor student, inundated with all of this information and generally sleep-deprived?*

Professor: *For the sleep deprivation, that's easy: sleep more (and party less). For understanding virtual memory, start with this: **every address generated by a user program is a virtual address**. The OS is just providing an illusion to each process, specifically that it has its own large and private memory; with some hardware help, the OS will turn these pretend virtual addresses into real physical addresses, and thus be able to locate the desired information.*

Student: *OK, I think I can remember that... (to self) every address from a user program is virtual, every address from a user program is virtual, every ...*

Professor: *What are you mumbling about?*

Student: *Oh nothing.... (awkward pause) ... Anyway, why does the OS want to provide this illusion again?*

Professor: *Mostly **ease of use**: the OS will give each program the view that it has a large contiguous **address space** to put its code and data into; thus, as a programmer, you never have to worry about things like "where should I store this variable?" because the virtual address space of the program is large and has lots of room for that sort of thing. Life, for a programmer, becomes much more tricky if you have to worry about fitting all of your code data into a small, crowded memory.*

Student: *Why else?*

Professor: *Well, **isolation** and **protection** are big deals, too. We don't want one errant program to be able to read, or worse, overwrite, some other program's memory, do we?*

Student: *Probably not. Unless it's a program written by someone you don't like.*

Professor: *Hmmm.... I think we might need to add a class on morals and ethics to your schedule for next semester. Perhaps OS class isn't getting the right message across.*

Student: *Maybe we should. But remember, it's not me who taught us that the proper OS response to errant process behavior is to kill the offending process!*

errant:
行为不当的