
Summary Dialogue on Concurrency

Professor: *So, does your head hurt now?*

Student: *(taking two Motrin tablets) Well, some. It's hard to think about all the ways threads can interleave.*

Professor: *Indeed it is. I am always amazed at how so few line of code, when concurrent execution is involved, can become nearly impossible to understand.*

Student: *Me too! It's kind of embarrassing, as a Computer Scientist, not to be able to make sense of five lines of code.*

Professor: *Oh, don't feel too badly. If you look through the first papers on concurrent algorithms, they are sometimes wrong! And the authors often professors!*

Student: *(gasps) Professors can be ... umm... wrong?*

Professor: *Yes, it is true. Though don't tell anybody – it's one of our trade secrets.*

Student: *I am sworn to secrecy. But if concurrent code is so hard to think about, and so hard to get right, how are we supposed to write correct concurrent code?*

Professor: *Well that is the real question, isn't it? I think it starts with a few simple things. First, keep it simple! Avoid complex interactions between threads, and use well-known and tried-and-true ways to manage*

thread interactions.

Student: *Like simple locking, and maybe a producer-consumer queue?*

Professor: *Exactly! Those are common paradigms, and you should be able to produce the working solutions given what you've learned. Second, only use concurrency when absolutely needed; avoid it if at all possible. There is nothing worse than premature optimization of a program.*

Student: *I see – why add threads if you don't need them?*

Professor: *Exactly. Third, if you really need parallelism, seek it in other simplified forms. For example, the Map-Reduce method for writing parallel data analysis code is an excellent example of achieving parallelism without having to handle any of the horrific complexities of locks, condition variables, and the other nasty things we've talked about.*

Student: *Map-Reduce, huh? Sounds interesting – I'll have to read more about it on my own.*

Professor: *Good! You should. In the end, you'll have to do a lot of that, as what we learn together can only serve as the barest introduction to the wealth of knowledge that is out there. Read, read, and read some more! And then try things out, write some code, and then write some more too. As Gladwell talks about in his book "Outliers", you need to put roughly 10,000 hours into something in order to become a real expert. You can't do that all inside of class time!*

Student: *Wow, I'm not sure if that is depressing, or uplifting. But I'll assume the latter, and get to work! Time to write some more concurrent code...*