ECE 252: Systems Programming and Concurrency Spring 2019

Lecture 23 — The Byzantine Generals Problem

Jeff Zarnett *2018-09-12*

Istanbul was Constantinople

Byzantium was an ancient Greek city, rebuilt by Constantine the Great under the name of Constantinople. It is now modern day Turkey. The word Byzantine refers to things relating to the empire based there, but in English has also come to mean a thing that is excessively complicated. But it's the last "extra" meaning that we want to talk about: characterized by deviousness. The Byzantine Empire was characterized by deviousness, alright: it was a place where generals and leaders would vie for power in underhanded ways, backstabbing and sabotaging others when they thought it was to their advantage. I guess it was very *Game of Thrones*, now that I think of it.

Where this crosses over into the realm of programming is in dealing with the unreliable nature of systems. It's not that we think that somewhere out there one of the processes in the system is malicious and trying to make the system fail... although that could happen... but our normal expectation is that the systems interacting may be faulty or have errors or bugs, not malicious intent.

However we get here, though, we still need a way to deal with the fact that not everyone agrees and sometimes messages get mixed up. And that is a communication problem, yes, but also a coordination problem as well. And if you've tried to get a group of five people to agree on where to go to lunch you may have found it unexpectedly difficult... now imagine that one person is (unintentionally or otherwise) trying to sabotage the planning so organizing lunch will fail.

The generic form of the Byzantine Generals problem looks something like this. There is a *General* who is giving the orders. There are also n *Lieutenants* who each are commanding a group of the Empire's troops. The individual troops just do what they're told so there's no further concern for their actions. The lieutenants can communicate with one another through messages, and they get their orders from the general. All of the lieutenants have to work together for their plan to be a success, otherwise there is chaos.

No problem, you might imagine. The general tells the lieutenants what to do and they do it! And if one of the lieutenants does the wrong thing, that lieutenant is a traitor. But it's not so simple, because the general can be disloyal too. Horrible Bosses: Byzantium!

Suppose you are a disloyal general. Your Emperor has commanded you to attack, but you want the attack to fail, but also seem like it's not your fault. What do you do? You can issue different orders to different lieutenants, they will be all confused and uncoordinated, and oh gee darn, I guess the attack didn't go as planned! And you can blame the lieutenants who are now dead for going too early, or something, because it's not like they can defend themselves now...

So a loyal general sends the same message to all lieutenants and a disloyal one sends different messages to different lieutenants. It's key to remember that "loyal" really means "functioning" and "disloyal" means "faulty" [HZMG15]. It's fun to make it seem like human actors are doing things for their own evil-villainy-related reasons, but it is just computer interaction that presumably has no actual malicious intent. It's also worth noting that being in one state or the other is not permanent: a functioning unit can be damaged, a malfunctioning one can be repaired, or a problem can be transient (i.e., some electromagnetic interference flipped a bit here or there) and resolve itself with no further action.

One of the key decisions is about how much disloyalty your system can tolerate. Given enough bad actors, things will go wrong. If everyone is disloyal, utter chaos will result. But is it enough to tolerate one disloyal participant? Two? The line will have to be drawn somewhere... But at the very least we don't want to let one disloyal individual ruin things for everyone, even if that disloyal individual is the boss.

Should I Stay or Should I Go Now?

In the examples that we will discuss there are two kinds of command: attack and retreat. In a real system the options don't have to be quite so binary (we could give any orders) but for the purpose of demonstration and coming to grips with this problem we'll use the binary example.

References

[HZMG15] Douglas Wilhelm Harder, Jeff Zarnett, Vajih Montaghami, and Allyson Giannikouris. *A Practical Introduction to Real-Time Systems for Undergraduate Engineering*. 2015. Online; version 0.15.08.17.