CS162 Note 2: OS Design Patterns (Extra)

Kevin Moy

Abstract

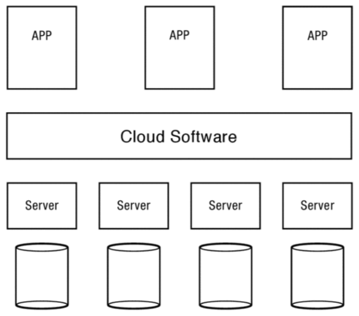
Remember that many other software systems have to handle multiple users/processes and run third-party programs- just like the OS does. So the common responsibilities we know, as well as their solutions: resource allocation, fault isolation, communication, hardware abstraction- are the same in the OS as in these other systems. We'll look at some of these implementations in this note.

1 Similar Systems

Let's take a look at some similar systems.

1.1 Cloud Computing

Cloud computing is a model where applications rely on warehouses and large-scale data centers for resources and computation instead of personal computers- meaning the user actually doesn't need to manage any of this stuff. Specifically, the cloud software does all the abstraction work of server resources from cloud applications:



The cloud software must take very similar OS-like responsibilities. It must determine how to allocate resources to the many cloud applications. It must fault-isolate these applications. We must make updates and improvements in cloud resources/computing *portable* to application development. Cloud software must also support communication between cloud applications.

1.2 Web Browsers

We know the main purpose of web browsers is to load and display the web pages we want. However, we also know a lot more goes on behind the scenes.

Many pages embed Javascript, whichs can be buggy or malicious. Like an OS, browsers must isolate this web page (should the bad script run) from *everything* else- other web pages, the user, and even the browser. Same goes for plug-ins and extensions.

We know browsers *definitely* have to handle parallel execution with multiple tabs. Browsers also have to make server changes- and crashes- transparent to the user. Browsers ALSO have to support *portability for scripts*: making sure the same scripts works across different operating systems and hardware.

1.3 Multiplayer games

Multiplayer games like Fortnite, Fallout, Grand Theft Auto, etc. often have *extensibility APIs*: software that allows third-party extensions like mods or DLCs. However, we also must make sure these extensions (just like in web browsers) don't actually create loopholes or bugs in the actual game. In the case that in-game objects are spread across multiple machines, we need to account for this with extension code. The game should also make it easier to develop extensions- since most successful games allow for a lot of these.

1.4 Multi-User Database Systems

Multi-user databases (Oracle, SQL Server, etc.) allow organizations to mess with a ton of company data. Large-scale data allows a lot of optimization for a lot of things, but we again have to take on responsibilities: simultaneous access from different locations, resource allocation among users, data privacy, and ensuring data is up-to-date on disk (periodically). Another thing the database managers have to account for is masking machine failures such that the actual data won't be affected.

1.5 Parallel Processes

With a multiprocessor computer, we can exploit **parallel processing**: using multiple processors to handle a portion of work from each application to maximize speed. However, when we do this, we must ensure that accesses to shared data is *synchronized* to preserve consistency and avoiding reads to data that isn't updated.

We can essentially build a mini-operating system on top of our normal one (like a VM!) to achieve user-level parallelism by handling assigning work to our multiple processors. This mini-OS has to decide things like priority of tasks, abstracting hardware physics from software, and allowing parallel programming with common data structures like trees and lists.

1.6 The Internet

The Internet *definitely* has some responsibilities to handle, given that like more than a billion use it AT THE SAME TIME. At the physical layer, all these billion users actually use the same finite resources- we obviously have to handle this. We have to also handle the vast amounts of malicious code that comes at it too. Should the Internet treat all users identically (NET NEUTRALITY) or should ISPs allow "premium access"? Can we *redesign* the Internet to prevent the vast array of cyberattacks that exist?

The Internet has to handle the case of limited size packets, limited distance to send these packets, and the chance that some of these packets could be corrupted along the way. So it'll provide the *illusion* that it can send packets from anywhere in the world to anywhere in the world.

Finally, obviously programming evolves and so do network applications over time. The Internet has to stay robust and supportive to these chances.

2 Summary

So we see many of the above systems that we're familiar with are actually *very similar* to operating systems- at least in terms of the responsibilities they need to take care of. Studying the operating system will allow us to understand much of how these other systems work!