Chapter 7: What's Next?

Introduction

And now, after writing a simple operating system kernel and learning the basics of creating kernels, the question is "What's Next?". Obviously, there is a lot to do after creating 539kernel and the most straightforward answers for our question are the basic well-known answers, such as: enabling user-space environment in your kernel, implementing virtual memory, providing graphical user interface or porting the kernel to another architecture. This list is a short list of what you can do next with your kernel.

Previously, I've introduced the term *kernelist* [1] in which I mean the person who works on designing an operating system kernels with modern innovative solutions to solve real-world problem. You can continue with your hobby kernel and implementing the well-known concepts of traditional operating systems that we have just mentioned a little of them, but if you want to create something that can be more useful and special than a traditional kernel, then I think you should consider playing the role of a kernelist.

I've already said that the kernelist doesn't need to propose her own solutions for the problems that she would like to solve. Instead of using the old well-known solutions, a kernelist searches for other better solutions for the given problem and designs an operating system kernel that uses these solutions. Scientific papers (papers for short) are the best place to find novel and innovative ideas that solve real-world problem, most probably, these ideas haven't been implemented or adopted by others yet[2].

In this chapter, I've chosen a bunch of scientific papers that propose new solutions for real-world problem and I'll show you a high-level overview of these solutions and my goal is to encourage the interested people to start looking to the scientific papers and use their solutions. Also, I would like to show how the researches on operating systems field[3] innovate clever solutions and get over the challenges. However, before starting with the chosen papers, the first section of this chapter discusses general concepts that are related to operating systems, we haven't discussed these concepts previously and they will be needed to make the papers that we are going to present easier to grasp. A small note should be mentioned, this chapter only shows an overview of each paper which means if you are really interesting on the problem and the solution the a given paper represents, then it's better to read it[4].

[1] In chapter where the distinction between a kernelist and implementer has been established.

[2] Scientific papers can be searched for through a dedicated search engine, for example, Google Scholar.

[3] Or simply the kernelists!

[4] It is easy to get a copy of any mentioned paper in this chapter, you just need to search for its title in Google Scholar (https://scholar.google.com/) and a link to a PDF will show for you.

In-Process Isolation

In current operating systems, any part of a process can read from and write to any place of the same process' memory. Consider a web browser which is an application like any other application consists of a number of different modules [5] and each one of them handle different functionality, rendering engine is one example of web browser's module which is responsible for parsing HTML and drawing the components of the page in front of the user. When an application is represented as a process, there will be no such distinction in the kernel's perspective, all application's modules are considered as one code that each part of it has the permission to do anything that any other code of the same process can do. For example, in web browser, the module that stores the list of web pages that you are visiting now is able to access the data that is stored by the module which handles your credit card number when you issue an online payment. As you can see, the first module is much less critical than the second one and unfortunately if an attacker can somehow hack the first module through an exploitable security bug, she will be able to read the data of the second module, that is, your credit card information and nothing is going to stop her. This happens due to the lack of *in-process isolation* in the current operating systems, that is, both sensitive and insensitive data of the same process are stored in the same address space and any part of the process code is permitted to access all these data, so, there is no difference in your web browser's process between the memory region which stores that titles of the pages and the region which stores you credit card information. A severe security bug known as *HeartBleed vulnerability* showed up due to the lack of in-process isolation, next, HeartBleed will be explained to show you how this real-world problem may impact our systems and then one of the solutions that has been proposed by kernelists will be discussed.

[5] In the perspective of programmers.