Operating Systems 2

Introduction

Arkadiusz Chrobot

Department of Information Systems

February 26, 2024

Outline

Literature

Primary Literature

1. Robert Love, *Linux Kernel Development*, Third Edition, Addison-Wesley, Upper Saddle River, NJ, 2010
2. Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman, *Linux Device Drivers*, O'Reilly, 2005
3. Wolfgang Mauerer, *Professional Linux Kernel Architecture*, Wiley Publishing, Inc., Indianapolis, 2008
4. Sreekrishnan Venkateswaran, *Essential Linux Device Drivers*, Prentice Hall, Upper Saddle River, 2008
5. Daniel P. Bovet, Marco Cesati, *Understanding the Linux Kernel, 3rd Edition*, O'Reilly Media, Sebastopol 2005

Literature

Additional Literature

1. Mel Gorman, *Understanding the Linux Virtual Memory Manager*, Pearson Education, Inc., Upper Saddle River, 2004
2. Maurice J.Bach, *The Design of the Unix Operating System*, Prentice Hall 1986

Literature

Webpages

1. Linux Weekly News
2. Linux Kernel Newbies
3. Linux Cross Reference
4. Embedded Linux and Kernel Engineering
5. Build own USB device on linux-based board! [en] - Krzysztof Opasiak
6. Linux Kernel Documentation

Linux History

The development of the Linux operating system (more specifically, the kernel) has been started in 1991 by a Finnish Computer Science student Linus Benedict Torvalds. But the story began a long time before that date ...

Linux History

- In 60. of the last century the MIT, General Electric and AT&T started working on an innovative, time-sharing computer system. The operating system for this computer, called MULTICS, was prepared by the MIT and AT&T.
- The project run into problems and the AT&T backed out. Ken Thompson who worked in the Bell Labs (part of the AT&T company) and was one of the programmers that wrote MULTICS started to write a game *Space Travel* on a PDP-7 computer. Then, in 1969, he began developing the Unix operating system (originally UNICS).
- Other programmers from AT&T gradually joined Thompson (like Dennis Ritchie, Malcolm Douglas McIlroy, Brian Kernighan, Rob Pike).
- Douglas McIlroy contributed the Unix pipelines.
- In 1973 the biggest part of the Unix source code was rewritten in the C language created by Denis Ritchie.

Linux History

- The AT&T was a monopoly in a telecommunication industry. They could not sell software, so they sold computers with pre-installed Unix, together with its source code.
- The system became very popular in academia and industry. The source code of Unix was used in operating systems courses at universities. John Lions, an Australian Computer Science researcher, wrote a textbook (so-called Lions Book) that contained parts of the Unix code for those classes.
- Researches at University of California in Berkeley issued their own version of Unix, that had sockets for network communication. One of them, Bill Joy, added virtual memory subsystem and several tools like `vi` and C-shell (`csh`).
- Many other universities and companies started to issue their own versions of Unix.
- To manage the variety of Unix implementations IEEE, ISO and The Open Group issued two standards POSIX and SUS.

Linux History

Short Brake

The Unix was successful because of the *Unix philosophy* — the way it was designed. The main features of this design are as follows:

- two main concepts — the process and the file,
- the KISS rule — "Keep It Simple, Stupid",
- small number of system calls (about 100),
- simple to use and fast system calls ("do one thing and do it well"),
- interprocess communication primitives,
- written in portable, high-level programming language.

Linux History

Continued

- In 80. of the previous century the US Government decided to divide AT&T into several companies. One of them overtook the software branch and started to licensing the Unix, also the already sold versions!
- The new license forbade using the Unix source code in operating systems classes at universities. The Lions Book become illegal!
- Andrew S. Tanenbaum, a professor of Computer Science at Vrije University at Amsterdam, upset by this situation developed MINIX an operating system for students that is Unix-like, i.e. its design follows the POSIX and SUS standards, but is based on microkernel architecture.
- Linus Torvalds, Finnish student of Computer Science, upset by the shortcomings of MINIX started to develop an operating system kernel, and he shared its source code through Usenet.
- Soon he was joined by other programmers. The user tools were provided by the GNU and other organizations.

Linux History

Concluded

Nowadays Linux is one of the widely used operating system (if not the most). The kernel has alone 27.8 million lines of code (as of January 2020). Even Ken Thompson uses Linux.

Linux Features

- It's a free (GNU GPL v. 2.0) operating system kernel.
- It's a Unix-like system.
- It's written mostly in the C language, but some parts are written in assembly language, which uses the AT&T notation.
- Since version 6.1, it is possible to add to the kernel code written in Rust.
- It is or was available for hardware based on ARM, AMD, Intel, POWER, PowerPC, MIPS, Sparc, UltraSparc and many other processors.
- It supports parallel architectures such as SMP and NUMA.
- It is provided together with user software, such as shells, X-Window, and so on. Such a collection of programs, including the kernel, is called a Linux distribution.

The Linux Kernel Overview

The kernel is a part of operating system that is always present in the RAM when the computer is on. It supervises devices and user processes. All kernel activities are performed in a system mode. The kernel also has its own set of addresses that it uses to access the memory. These addresses, the memory they reference and the state of the kernel are called a *kernel-space*. By contrast the state of user processes together with the content of their memory and the addresses that these processes use to access the memory are called a *user-space*. The kernel actively performs operations only when it is initialized, i.e. when a system boots up. After that it becomes an *event-driven* software. These events may be initiated by user processes, when using a *system call*. The *system call* is a special kernel function that can be invoked by a user process. Usually it is not called directly, but with the use of some of the functions from the standard C language library. Some of these function don't invoke any system calls, but most of them use at least one system call.

The Linux Kernel Overview

The system call is performed in the kernel-space and it has an access to all data about the user process that invoked it, thus it is performed in the *process context.* The event that the kernel reacts to can be initiated by a hardware. Such events are called *interrupts* and they are asynchronous to the kernel activities. It means they can interrupt some important operations, thus they have to be served quickly. Because of that, the kernel code (the so-called *interrupt handler*) that handles the interrupt runs in a special *interrupt context*, which is not associated with any user process. The Figure 1 summarises the concepts of the user-space, the kernel-space, the process context and the interrupt context.

The Linux Kernel Overview

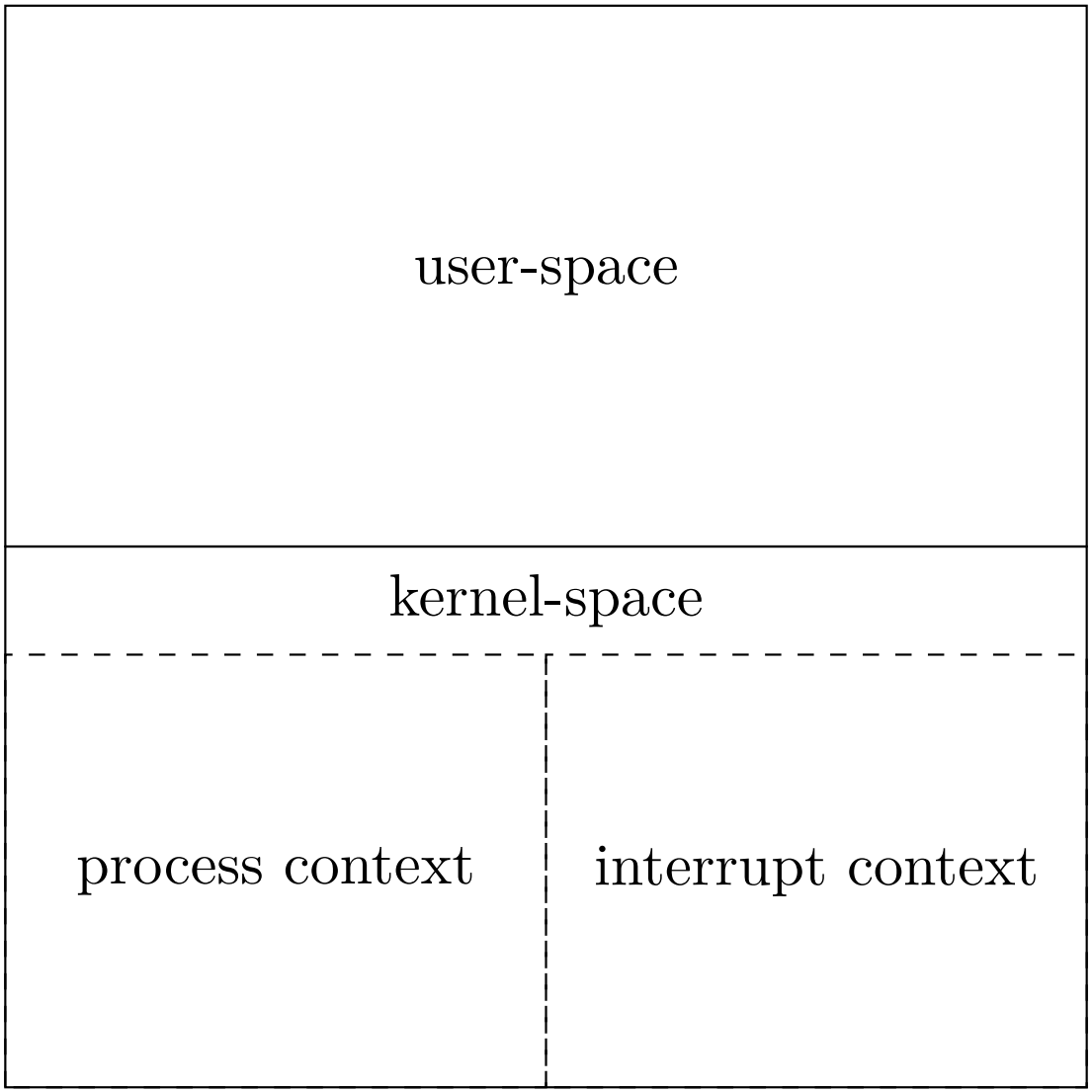

Figure 1: The main concepts of kernel

Linux Kernel vs. Other Unices Kernels

- Linux just, like Unix is a monolithic kernel.
- It supports dynamically loaded kernel modules.
- It supports multiprocessor parallel architectures like SMP and NUMA.
- It supports kernel threads preempting starting with the version 2.6.
- It doesn't swap kernel pages, only the user processes pages.
- It doesn't support streams, which are in Linux implemented totally in the user-space.

Linux Kernel Versioning

At the beginning the consecutive versions of the Linux kernel were marked with three numbers separated by a dot, like this: 2.2.1. The first number is the so-called *major version*, and the second is the *minor version*. For the development releases the minor number was odd, and for the stable releases it was even. The former were usually undergoing significant changes of code and they were not recommended for production environments. The latter contained fixes of discovered bugs and their code was not changing significantly. The last number, called *revision number* just indicates the order of releases. The version numbering changed with the release of the 2.6 kernel. The kernel programmers decided that it was so well designed, that no development version was needed. All significant changes were performed on the stable version. This led to the necessity of using a forth number that indicated that the version contains fixes of significant bugs.

Linux Kernel Versioning

With the release of the 3.0 version the numbering changed once again. The minor number is just used for indicating subsequent kernel releases. The release candidate versions are designated with the `-rc` suffix. There are also some long term support version. This lecture and the laboratory classes are mostly about the 4.15 version supported by Canonical. The newest stable kernel version, as of February 2024, is 6.7.6.

Introduction to Kernel-Space Programming

- The majority of the kernel code is written in the C language, only small parts are written in Rust and the assembly language.
- The kernel programmers often use the infamous `goto` statement, usually for optimization and exception handling.
- The standard C language library in unavailable in the kernel-space. Instead the kernel has its own header files and replacements of the major C functions. For example, instead of the `printf()` function the `printk()` function is used. The string processing functions from the user-space have their equivalents in the kernel-space.
- The kernel code has to be portable, which means that the kernel programmers have to, for example, take into consideration the order of the bits and bytes (the endianness).
- The concurrency and synchronization issues are always present in the kernel-space.

Introduction to Kernel-Space Programming

- The kernel allocates for every user-space process a kernel-space call stack (Do not confuse it with the user-space call stack!), which is used by system calls and kernel functions. **Its size is limited only to two pages!**
- The floating-point operations in kernel space are not directly supported. Fortunately, they are usually not needed.
- There is almost no memory protection inside the kernel-space.
- The kernel code uses GNU C extensions, like the *inline* functions, the *inline assembly* and the `likely()` and `unlikely()` directives for annotating the conditions of conditional statements.

Questions

?

The End

Thank You for Your attention!