Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Operating Systems

Part 1: Virtualization – 0) Introduction

Revision: `master@d08e360 (20220915-171922)`

BTI1341 / Fall 2022-23

P. Mainini / E. Benoist / C. Fuhrer / L. Ith

Outline

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Welcome!

Introduction

- Emmanuel Benoist
  - email: `emmanuel.benoist(at)bfh.ch`
  - Office: N671 (Villa Security)
  - “Acronym”: BIE1
  - Homepage: `http://www.benoist.ch/`
- PhD at the University of Caen (France)
- Professor at the Berner Fachhochschule
  - Teach Computer Science in Biel since 1999
  - Specialties: Algorithmic, Web Programming and Web Security
- Web Security and Privacy protection on the Web
  - Member of the Institute for Cybersecurity and Engineering (ICE)
    - Web Security
    - Privacy Protection on the Web
    - “Identity” in a broad sens

Course Contents

This course is broadly organized into three parts:

1. **Virtualization**
   - Processes
   - Scheduling
   - Memory
2. **Concurrency**
   - Threads
   - Synchronization
3. **Persistence**
   - I/O
   - File systems

We will also meet some auxiliary topics on our way...

Learning Objectives

After this course, you should have gained a more thorough understanding of operating systems. Especially:

- Know what an operating system is and the main functionalities it offers
- Understand processes, threads, memory management, I/O and file systems
- Be aware of common synchronization issues and know mechanisms for their resolution

More informally: have a good starting point to explore operating systems on your own and enjoy their intricacies!

Prerequisites

For the best experience, you should bring with you:

- A fundamental understanding of computer architecture (things like CPU, RAM, storage, I/O devices etc.)
- Understanding of the C programming language, as taught in computer science basics courses
- A running GNU/Linux operating system (virtual machine is OK!)
- Interest and motivation to play around with things

Course Organization

The course and accompanying material can be obtained on Moodle and Gitlab. Addresses are given separately.

Grading

BTI1341 is a **Pb** module, a total of **100 Points** can be achieved:

- **25 Points in a practical lab**
  - Done during the semester
  - Programming in C
  - Two parts:
    Memory virtualization
    Concurrency
- **75 Points at the final exam**
  - Takes place during the exam weeks, duration 120 minutes
  - Contents: all topics treated in the course, except those marked as optional (“ ★ ”)
  - You can bring *2 double-sided pages (A4)* of notes
  - No other material is allowed, especially no electronic devices

Notation

Throughout the course slides, we'll use some notation to highlight things:

☞ **Hint:** provides useful information which may help understand or do things.

★ **Optional task or topic:** Denotes an interesting topic recommended for further self-study, not part of the exam.

⚠ **Warning:** Indicates something important.

Literature

This course is based on the book:

Operating Systems: Three Easy Pieces (*OSTEP* for short) by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. ([ADAD18])

It is freely available online at `http://ostep.org`!

No other textbook is required for following this course, additional material will be provided or referred to where required.

Not having heard something is not as good as having heard it; having heard it is not as good as having seen it; having seen it is not as good as knowing it; knowing it is not as good as putting it into practice.

—*Xunzi*

Introduction to Operating Systems

- Which OS do you know?
- What is an OS?
- What functionalities does it offer?
- Do we need one? Why?

What is a Program?

Basically *just instructions* (billions!), in *memory*, executed step-by-step by the *processor (CPU)*.[1]

1. The processor fetches an instruction from memory
2. It then decodes it to know what to do...
3. ...and executes it afterwards
4. After this, the next instruction is fetched and executed (back to step 1)

Eventually, when all instructions have been processed, the program ends. (does it?)

[1] This is a simplification of what we call the *Von Neumann model of computing* ★

Program Instructions

```
1125:     55                   push   rbp
1126:     48 89 e5             mov    rbp,rsp
1129:     b8 00 00 00 00       mov    eax,0x0
112e:     5d                   pop    rbp
112f:     c3                   ret
```

Resource Management

Now that we know what a program is, and how it is executed...

- Who decides *which* program is to be run?
- How can we have *multiple* programs running at the same time?
- If so – which one is the first?
- Which program may access a *resource* (e.g. the screen or a USB stick)?
- How is it ensured that all programs play nicely with each other?

The answer to these questions (and more of them) lies in the operating system...

Virtualization

One of the key concepts in operating systems is virtualization :

- A program only sees a *representation* of a (physical) resource, e.g. the CPU or memory
- Often, this representation is more general, powerful and easy-to-use
- Each running program gets its own set of virtualized resources
- The program does not notice (in general) that the accessed resources are shared

The OS then acts as a resource manager , ensuring availability and fair distribution.

Virtualization: CPU

When virtualizing the CPU, each program has the impression to run on its own CPU:

- Enables running multiple programs at the "same" time
- Provides the illusion of having an infinite number of CPUs
- Instructions from different programs do not interfere with each other

➠ **Demo:** `ostep-code/intro/cpu.c`

Virtualization: Memory

With memory virtualization, each program has the impression to access its own *address space*.

- The same (virtual) memory addresses can be used by different programs
- Size and layout of memory must not necessarily be the same as for the real, physical memory of the machine
- Memory of different programs is isolated from each other

➠ **Demo:** `ostep-code/intro/mem.c`

Interlude: ASLR

Modern operating systems implement *address space layout randomization (ASLR)*: ★

- Executable code, loaded libraries etc. have different memory addresses on every program execution
- Introduced as a security measure against various types of attacks
- May hinder OS exploration and debugging

☞ To temporarily disable ASLR when running a program (GNU/Linux), use:

```
setarch `uname -m` -R <executable>
# e.g. setarch `uname -m` -R /bin/bash
```

Concurrency

Another important topic is concurrency:

- Initially, concurrent things happened mostly in the OS (why?)
- Concurrency leads to specific issues which need to be addressed
- Modern, multi-threaded programs now also have the same issues
- Synchronization methods are an important topic to be treated

➠ **Demo:** `ostep-code/intro/threads.c`

Persistence

The last major topic in this course is persistence :

- Data stored in memory is volatile and lost when the system is powered off
- I/O devices such as hard drives and solid-state drives (SSDs) are used for permanent storage
- Access to these devices is managed using directories and files in a *file system (FS)*
- In general, I/O devices are not virtualized but shared between applications

➠ **Demo:** `ostep-code/intro/io.c`

OS Design Goals

Having introduced virtualization, concurrency and persistence, we will now highlight some goals in *operating systems design*:

Abstraction
: The system should be easy and convenient to use; introducing abstractions enables this.

Performance
: Overhead, both in space (memory/disk) and time (CPU), needs to be kept small

Protection
: The OS, as well as individual programs require protection from each other. One of the main principles used here is *isolation*.

Reliability
: When the operating system fails, all applications fail as well. An OS must be capable of running 24h/day.

Security
: Protection against malicious programs, attackers. Can be considered as an extension of protection.

Efficiency
: Applies to various areas; with mobile and internet of things (IoT) devices, *energy efficiency* is critical.

Appendix

Bibliography

[ADAD18] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 1.00 ed., Arpaci-Dusseau Books, August 2018, Available online: `http://ostep.org`.