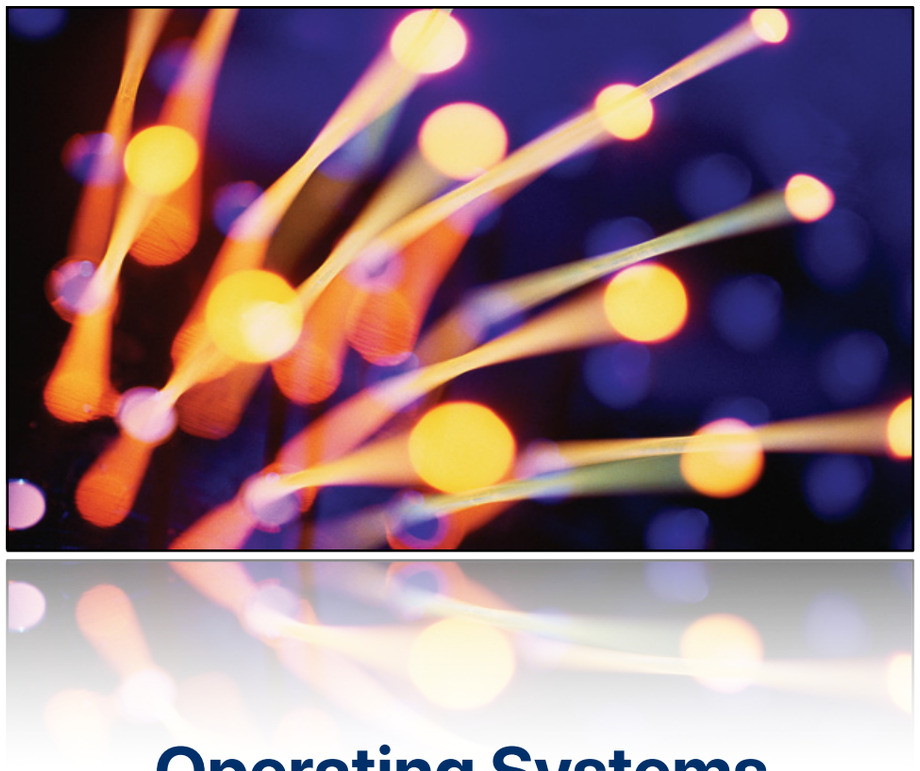
Introduction



Operating Systems

Baochun Li
University of Toronto

About me

Career

- Assistant Professor (2000)
- Associate Professor (2005)
- Professor (2008)
- IEEE Fellow (2015)

Research

- I lead a research group with around 8 grad students
- Cloud computing
- Networking
- Distributed systems

http://iqua.ece.toronto.edu/bli/ (or just google Baochun)

Two objectives of the course

Understand basic concepts of operating systems

- Fundamental principles
- Processes and threads
- Thread synchronization
- CPU scheduling
- Memory management
- File and I/O systems
- Virtual machines (and time permitting, security)

Gain practical hands-on experience

- Using the BLITZ system in our labs
- Reinforce what you have learned from lectures

Labs

We use the BLITZ system in our labs

BLITZ includes a CPU emulator, assembler, compiler for a high-level kernel programming language (called **KPL**), and a debugging system

It is simple, yet realistic enough to understand how things work

You will read, understand, and write real operating systems code!

A few words about the labs: BLITZ

BLITZ includes —

- A CPU architecture, emulated in software as a virtual machine
  - Similar to RISC CPUs, such as MIPS, SPARC and PowerPC
  - Emulation runs on the host machine (such as Linux)
  - Basic OS code to be extended in each project

Developing on BLITZ

- Edit, compile, link on the host machine
- Execute on the virtual machine (emulator)
- Debugging tools are provided in the emulator

Why BLITZ?

One extreme: directly work on x86 hardware

- Modifications to the Linux kernel
- Stanford Univ used an actual x86 virtual machine
- Difficult to develop and debug

The other extreme: no hardware exists in OS labs

- Example: simulate a CPU scheduling algorithm

Neither extremes are good choices

- There is no substitute to learning on a real OS kernel based on hardware
- **Best**: emulated hardware architectures — Nachos used MIPS
- But debugging with C/C++ can be time-consuming

Installing BLITZ on your own computer

We will provide a **Docker container image** for each lab

You just need to install Docker to run this image

It's like a virtual machine, just faster

It contains a Linux OS and all the required tools from BLITZ precompiled

We will provide more detailed instructions in Lab 1

Grading

Labs: 60% (20% each)

Lab 1 will not be marked

Course Project: 40%

Textbooks and lecture notes

***Required:* Operating Systems: Three Easy Pieces**, by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau

Free! (but paper copies are also available for purchase online)

Available for download at **http://www.ostep.org**

Chapters and sections covered will be available in the lecture notes

Lecture notes are made available in PDFs

Other reference textbooks

Operating System Concepts, A. Silberschatz, P. Galvin, G. Gagne, 10th Ed., Wiley, 2018 (or 8th Ed., 2008; 9th Ed., 2012).

Principles of Computer System Design, An Introduction, J. H. Saltzer, M. F. Kaashoek, Morgan Kaufmann, 2009.

Additional required reading

The BLITZ Documentation

- PDFs are available for download from the course website today (as a .zip archive)
- Source code for all BLITZ tools are also available for download

Reading BLITZ documentation

Required reading:

An Overview of the BLITZ System (7 pages)

Overview of the BLITZ Computer Hardware (8 pages)

The BLITZ Architecture (67 pages)

The BLITZ Emulator (46 pages)

Overview of KPL (66 pages)

A lot of pages — but well worth your time!

Course website

Official course website: oscourse.org

All course-related information is available

Course syllabus, BLITZ documentation, labs, lecture notes

WeChat group — online discussions only

Getting started

Baochun Li, Department of Electrical and Computer Engineering, University of Toronto

What is an operating system?

Windows?

MacOS

Linux

Smartphone operating systems

iOS

Android

Operating systems: a narrow view

User programs are not considered as part of an operating system

Examples —

The graphical user interface (GUI)

Productivity applications (such as the Windows Explorer)

Main distinctions

Operating systems live a long time, since they are hard to write

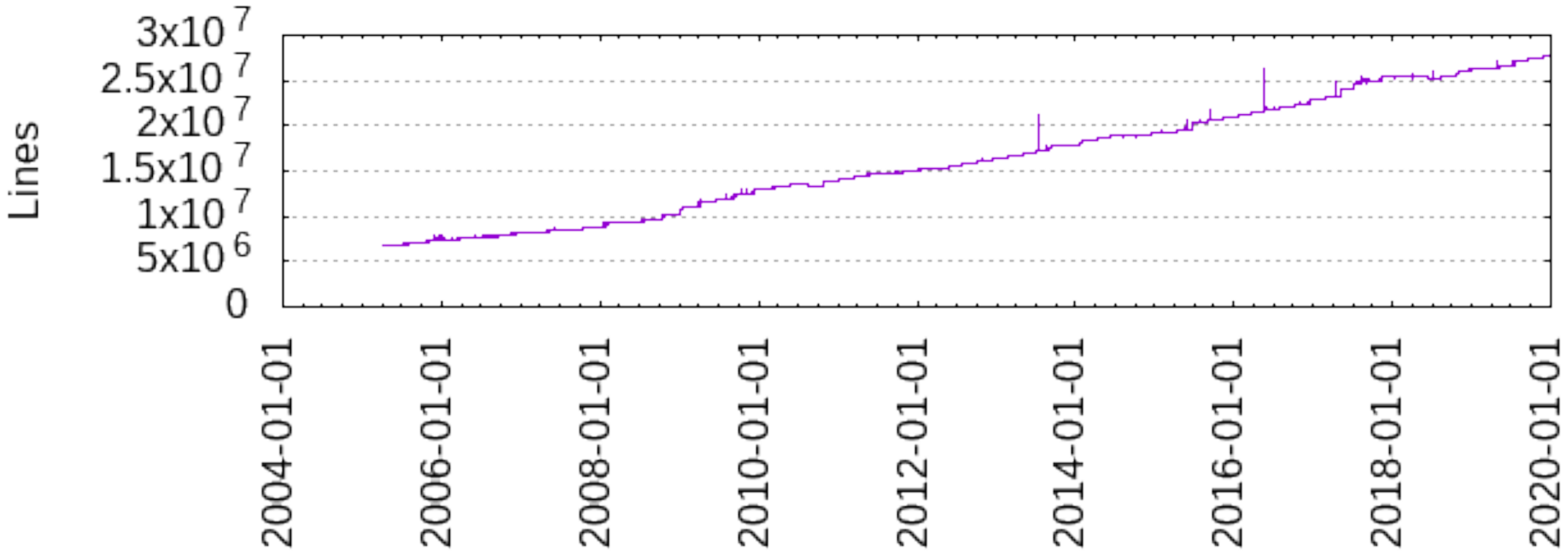
User programs offer choices to users (e.g., a web browser)

Operating systems are complex

The Linux kernel has a total of 27,852,148 lines of code as of January 1, 2020

66,492 files (source: https://phoronix.com/misc/linux-eoy2019/index.html)

In 2019 alone, there were 74,754 commits to the Linux kernel tree, adding 3,386,347 lines of new code and 1,696,620 lines of code were removed



Operating systems:

A brief history

Baochun Li, Department of Electrical and Computer Engineering, University of Toronto

Batch Systems

Example: IBM System/360 and OS/360 (1960s)

Main innovation: Multiprogramming across "jobs"

Partition the memory into several pieces to fit different jobs

While one job was waiting for I/O to complete, another job could be using the CPU

Timesharing Systems

Batch systems do not allow a quick turnaround time for users

Timesharing systems allocate CPU to provide fast, interactive service to a number of users, and work on batch jobs in the background

Example: MULTICS (1965-1974)

The concept of a “computer utility” just like electricity

MIT, Bell Labs, General Electric

Not successful — goals were too ambitious at the time

Similar to “cloud computing” today

Minicomputers and UNIX

DEC PDP-1 (1961): 4K of 18-bit words for $120K

Ken Thompson designed a stripped-down, one user version of MULTICS, called UNIX, on a PDP-7

Over five decades, UNIX had many variants

Linux came from Linus Torvalds' work on MINIX, an educational UNIX operating system

AT&T System V: led to Sun Solaris, HP/UX, IBM AIX

BSD: led to FreeBSD and Mach

Mach led to MacOS

Personal computers

Gary Kildall: CP/M on Intel 8080 and Zilog Z80 (1974-1980)

Microsoft: MS-DOS on IBM PC (1981)

Apple: macOS (since 1984)

Microsoft: Windows NT (1996)

Windows NT 4.0 -> Windows 2000 -> Windows XP -> Windows Vista/7/8/10

But what is an operating system anyway?

At a high level —

A layer of software between applications and hardware

Implements an application programming interface (API) so that applications can access hardware

Manages resources that applications share

Layering

Layering: a fundamental principle

Layering is a design philosophy that provides a way to simplify the design of a complex system

The OS, as a software layer, is a good example

With layering, we need to make decisions —

Which function is to be implemented in which layer?

Software vs. hardware implementation?

Concerns: cost, performance, flexibility, convenience, and usage patterns — a tradeoff must be made

The operating system layer usually allows a *layer bypass*

Layer Bypass

Rather than completely hiding the lower, hardware layer, an operating system usually hides only a few features of the hardware layer

- Particularly dangerous instructions

The remaining features of the hardware layer, including most of the instruction set, pass through for use directly by the application layer

- For performance reasons

Reading

Required: Three Easy Pieces

Chapter 2: Introduction to Operating Systems

Recommended: Operating Systems Concepts

Chapter 1.1: What Operating Systems Do