UiT

THE ARCTIC
UNIVERSITY
OF NORWAY

Introduction

inf-2201 Operating System Fundamentals

Spring 2017

Lars Ailo Bongo (larsab@cs.uit.no)

UNIVERSITY OF TROMSØ

Windows 10

macOS

MINIX 3



iOS


TIZEN

free
RTOS

SAILFISH OS

FreeBSD

OpenBSD

NetBSD®

ESXi

ARDUINO

- Multics
- IBM OS/360
- IBM VM/370
- MS-DOS
- Unix’es
- Mach
- L4
- Exokernel
- …

- Barrelfish
- Corey/ The Library OS
- Tesselation
- Vortex
- …

- Application
- **Operating System**
- Hardware

- OS'es for many different types of devices
  - Differing requirements (functionality, footprint, real-time)
- Abstractions and many design issues are still shared

THE CRUX OF THE PROBLEM: HOW TO VIRTUALIZE RESOURCES

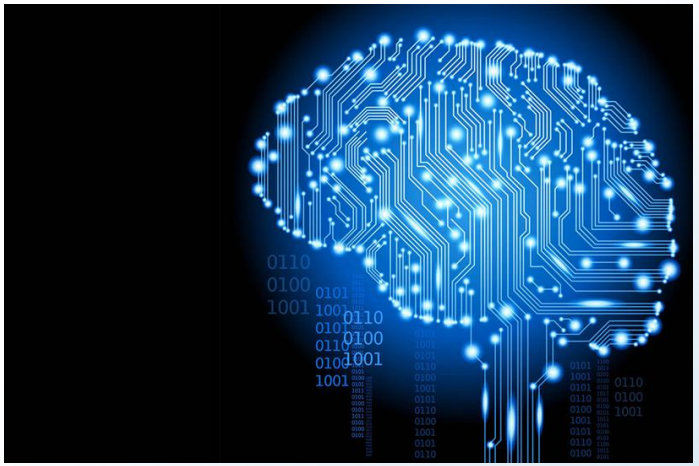
One central question we will answer in this book is quite simple: how does the operating system virtualize resources? This is the crux of our problem. *Why* the OS does this is not the main question, as the answer should be obvious: it makes the system easier to use. Thus, we focus on the *how*: what mechanisms and policies are implemented by the OS to attain virtualization? How does the OS do so efficiently? What hardware support is needed?

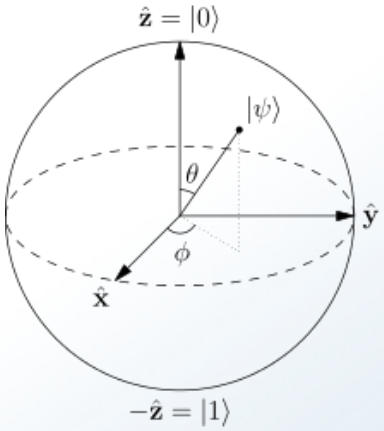
http://pages.cs.wisc.edu/~remzi/OSTEP/

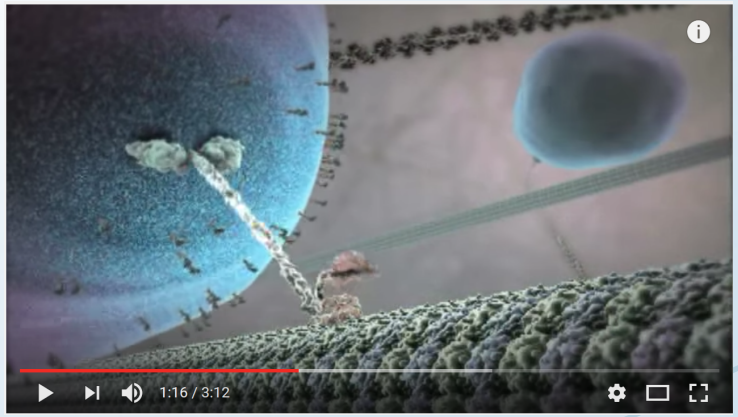
New challenges

- 1000s of cores
- TB’s of memory
- 100’000s of machines
- GPU, accelerators
- Power usage
- Sensors
- …

Future OS'es?




$\hat{\mathbf{z}} = |0\rangle$
$|\psi\rangle$
θ
$\hat{\mathbf{y}}$
ϕ
$\hat{\mathbf{x}}$
$-\hat{\mathbf{z}} = |1\rangle$


1:16 / 3:12

Outline

- How?
- Why?
- What?

Teaching staff

- Associate Prof. Lars Ailo Bongo
- Associate Prof. Tore Brox-Larsen
- Teaching Assistant Marius Wiik
- Teaching Assistant Nikolai Magnussen
- Our external sensor

Resources

- Web page: http://www.cs.uit.no/kursinfo/inf2201
  - Lecture plan, including readings and slides
  - Project plan
- Mailing list: inf-2201-s17@list.uit.no
  - Important information
- Slack team
  - Discussions
- GitHub organization: https://github.com/uit-inf-2201-s17
  - Project pre-code
  - Your solutions
  - Issues
- Wiseflow
  - Exams
- We will not use Fronter

TODO list 1:

1. Make sure you are subscribed to the mailing list
   - https://list.uit.no/sympa/info/inf-2201-s17
   - Can use non-UiT email
2. Create GitHub account
3. Request membership to GitHub organization
4. Get invite to Slack team

Why Study Operating Systems

- Understand how computers work under the hood
  - "You need to understand the system at all abstraction levels or you don't" (Yale Patt, private communications)
  - "The devil is in the details" (Unknown)
- Magic to provide infinite CPUs, memory, devices, and networked computing.
- Tradeoffs between performance and functionality, division of labor between HW and SW
- Combine language, hardware, data structures, algorithms, money, art, luck, and hate/love
- *And operating systems are key components in many systems*

How to Study Operating Systems?

- Read text
- Smaller exercises using existing operating systems
- Modifications to exisiting systems
  - Emulator
    - NachOS
  - ”Metal”, bare machine
    - Unix, Linux
    - Minix
- Develop your own! Gain experience on fundamental issues

Why Build a Real OS Kernel?

- Hear and forget (Paper approach)
- See and remember (Exercise and Modification approaches)
- Do and understand (Roll your own approach)
- Overcome the barrier, dive into the system
- Gain confidence: *You* have the power, not the SW, OS and computer vendors ☺

Our Approach

- 6 projects, all mandatory
  - From boot to a usefull OS kernel w/demand paging & file system
  - We hand out templates (*pre files*), but never the finished source (*post files*)
    - New bugs discovered every time the course is given ☺
- Lectures and Projects are (almost) synchronized
- Design Review during first week of each project
- Linux, C, assembler
- Close to the computer, but bochs useful to reduce the number of reboots

Project OS History

- **LurOS**
  - Stein Krogdahl, OS course, Dept. of computer science, UiTø, ca. 1978
  - Paper, but detailed
- **Mymux** (Mycron Multiplexer)
  - Stein Gjessing (1979?), later implemented and reworked by Otto Anshus (1981), OS course, Dept. of computer science, UiTø, around 1981-82-83
  - Mycron 1 (64KILObyte RAM, no disk, 16 bit address space, Intel 8080/Zilog 80, Hoare monitors, flermaskin (3, UART, 300 bits/sec, transparent process and monitor location, process and monitor migration between machines)
- **POS** (Project Operating System), a.k.a. **TeachOS**, a.k.a. **LearnOS**
  - Otto Anshus, Tore Larsen, first implementation by Åge Kvalnes (Brian Vinter), OS course, Dept. of computer science, UiTø, 1994-1998
    - Notebooks, Intel 486/Pentium
  - Princeton University, USA
    - Kai Li (1998), adopts and enhances the projects
  - Tromsø & Princeton
    - D240/COS 318, Kai Li, Otto Anshus, 1999
  - Tromsø/Princeton/Oslo
    - Inf-2201/COS318/INF242, 2001, Vera Goebel, Thomas Plagemann, Otto Anshus
  - Yale University

Course Approach

- You will be building your own operating system
- You will do it in steps
- For each step
  - We'll define what your OS should achieve for this step
  - We'll provide you with a starting point (code)
    - *You may well choose to use your own starting point*
  - You will contemplate a design and present a brief design report indicating design issues, discussions, and decisions. The design report is presented, discussed, and reviewed by staff
  - You develop and implement your solution. The solution is reviewed etc.
- For each step you will sweat
- By early June you will be ”King of the hill!”

King of the Hill

Literature

- *Modern Operating Systems 4/E*, by Andrew (Andy) Tanenbaum, Prentice-Hall, 2015
- All information given on the course web pages. The links provided are mandatory readings to the extent they are relevant to the projects
- We will also provide additional readings. Please, check the syllabus
- All lectures, lecture notes, precept notes and topics notes
- All projects
- Other books that may help you are:
  - *Protected Mode Software Architecture*, by Tom Shanley, MindShare, Inc. 1996. This book rehashes several on-line manuals by Intel
  - *Undocumented PC*, 2nd Edition, by Frank Van Gilluwe, Addison-Wesley Developers Press, 1997
  - *The C Programming Language*, Brian W. Kerningham, Dennis M. Ritchie
- **Note! Lectures covers important topics that are not addressed by projects**

Cooperation Policy: Working with your fellow students

- Discuss all concepts, techniques, technologies, and peculiarities with your fellow students
- But don’t in any way share your own code or copy code that is not developed by you.

Learning from Doing: Each student develops his/her own code

INF-2201 har en eksamensordning hvor oppgaver i kurset vurderes og gis veiledende karakterer som ved kursets slutt danner utgangspunkt for den endelige fastsettingen av karakter for kurset. På grunn av dette vil følgende bli betraktet som fusk iht. Forskrift for eksamener ved Universitetet i Tromsø:
Deling av kode (både i elektronisk og annen form) utviklet i forbindelse med oppgavene i kurset.
Kopiering av hele eller deler av løsning på oppgavene utviklet av andre.

Av hensyn til gjennomføring og karaktersetting for kurset i framtiden må kode og kursmateriale ikke på noe tidspunkt deles eller distribueres til andre enn faglig og administrativt personale ved UiT med ansvar for gjennomføring av kurset.

Med begrunnelse i det ovennevnte må alle studenter fylle ut og skrive under på følgende erklæring og levere denne til fakultetsadministrasjonen sammen med innleveringen av den første obligatoriske oppgaven.

Cheating

- Jan Fuglesteg

Grading Policy

- "Mappe evaluering":
  - 3 individual grades: combined to give final grade
  - No written exam
- Wiseflow
  - More details TBA
- **Note! It is your responsibility to make sure you submit the correct assignment on time**

Grading Mechanism

- 100 point scale (A is 90-100)
- Extra credits are additional to 100 ordinary point
- You can get an A without doing extra credits
  - But hard in practice
- Examiners follow grading guidelines:
  - Each part of assignment gives X points
  - Most points are for code correctness
  - Some are for code readability (comments, structure)
  - A few are for documentation
- Examiners may give short explanation for grades < A
- You should discuss your code/ solution with TAs

TODO list 2

- Read MOS 1.1-1.5
- Learn GNU assembly syntax
- Learn to use git/ GitHub