INF 3150/3160 Operating Systems

Fall 2003

Tore Larsen

Adaptions and additions to notes developed by
Otto J. Anshus

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
- In December you will be "King of the hill!"

People

- Carsten Griwodz, Ifi, UiO
- Pål Halvorsen, Ifi, UiO
- Tore Larsen, Ifi, UiTø & UiO
- Teaching Assistants (TA's)
- Faculty currently on sabbatical
  - Otto J. Anshus, Ifi/UiTø & Ifi/UiO
  - Vera Goebel, Ifi, UiO & UiTø
  - Thomas Plageman, Ifi, UiO

Topics

- Course approach
- Presentation of teachers
- Where do we find operating systems?
- What are the basic mechanisms for combining the operating system and other SW?
- Why do we study operating systems?
- … and how do we do it?

Is it challenging to write an OS?

- Yes, but you'll manage. Employing your own efforts, and the assistance of fellow students, TA's, and professors.
- Low-level, architecture dependent programming
  - We stick with only one architecture in only one configuration
- Race-conditions
- Let's see (next slide) what a great computer scientist experienced some time ago …

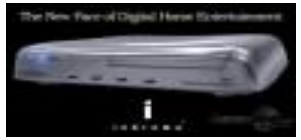
From Tony Hoare's Biography

http://research.microsoft.com/~thoare/

- …He led a team (including his later wife Jill) in the design and delivery of the first commercial compiler for the programming language Algol 60. (1960)
- …He then led a larger team on a disastrous project to implement an operating system
- …His research goal was to understand why operating systems were so much more difficult than compilers, and to see if advances in programming theory and languages could help with the problems of concurrency. (Queens Univ. 1968)

Where do we find OS'es these days?

Game boxes have OS'es?



Sidebar
Penetration of Microprocessors

- Early Nintendo game consoles (previous page, early eighties) was the first ever microprocessor-based fad. When Yngvar Lundh (left) witnessed this, he predicted that microprocessors would become as widely applied and as ”invisible” as electrical motors were at that time.
- More by Lundh at:
  - http://www.ifi.uio.no/ansatte/yngvar.html

Cell-phones have OS’es?

- Symbian
- Microsoft

Cameras have OS’es?

- FlashPoint
  - Pentax, Olympus, Sanyo, Kodak
- Sony?

Cameras also have software for exposure and focus

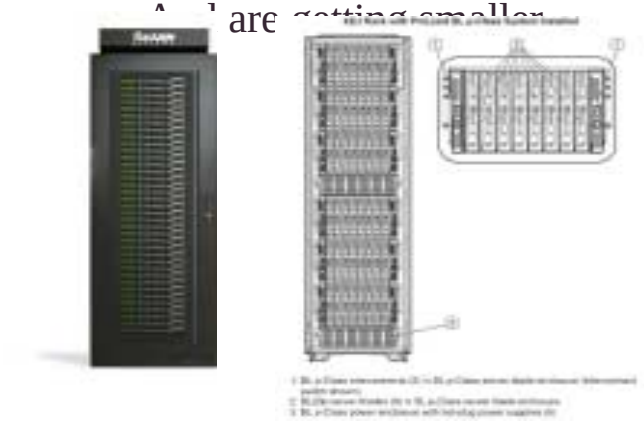
Cars have OS’es?

- Car manufacturer
- Controller vendor
- Software house
- Microsoft
- Real-time
  - Suspension
  - Engine
- Non real-time
  - Entertainment

Set-top boxes have OS'es?

- Seen in late nineties as possibly controlling gateway into computerized future home
- Fierce early competition
- Expectations not yet met

Big engines have OS'es



Observations and Two Questions

- OS'es for many different types of devices
  - Differing requirements (functionality, footprint, real-time)
- Abstractions and many design issues are still shared
- Does one size fit all, or is that a silly dream?
  - Rick Rashid (Mach ++) reportedly wanted to see Mach in light switches (which are currently being computerized)

OS, other programs, and HW

- OS and other programs use the hardware (processor, memory, input, and output)
- How is the OS's interaction with HW different from any other program?
- Let's look at some early HW:

Did this machine have an OS?

EDSAC, First operational Von Neuman Computer. Pictured 1949

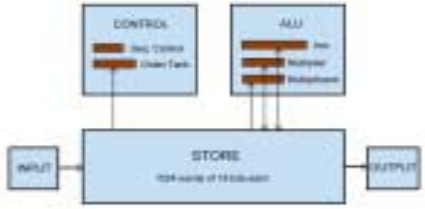
http://www.dcs.warwick.ac.uk/~edsac/

http://www.cl.cam.ac.uk/UoCCL/misc/EDSAC99/

EDSAC

- Project led by Maurice Wilkes at Cambridge
- Based on design written down by John Von Neumann (Neumann Janos)
- First operational Von Neumann machine
- Stored program concept
- Also
  - Subroutines
  - Bootstrap (shadow ROM BIOS)
    - 31 instructions

EDSAC Architecture



- Most architectures since have been like this
- Let's depart, use the blackboard, and revisit how this architecture proceeds, and how it connects with the OS

Proceeding through Instruction Stream

- Fetch instruction
- Decode instruction
- Fetch operands
- Execute
- Write back result
- Next instruction

- When decoding instruction, what to do if bit-pattern doesn't represent a (legal) instruction?
  - Halt? *No!*
  - Instead: Trap: fetch next instruction at predetermined address in memory. Make sure that you have placed your OS code there beforehand

Two Execution Modes

- Machine is in ”supervisor” (or similar term) -mode or privilege level
  - The full instruction set of the processor decodes
- Machine is in ”user” (or similar term) –mode or privilege level
  - Only a restricted set of instructions decodes, all other instructions, typically referred to as ”privileged instructions” are treated as ”illegal,” – decoding them causes a trap to the OS
- Instruction set designer must make sure that ”the right set” of instructions are privileged.
- *What about the instruction to set privilege level?*

How we may proceed

- OS starting user program
  - Load program and initialize
  - Set privilege level ”user”
  - Load instruction register from start of program
- User program requesting OS service
  - Make a mark ”somewhere” indicating which service is requested
  - Execute instruction that i bound to trap in decode
- *Still need mechanism that allows OS to ”preempt” user process, OS needs to be activated independently of running program. That can only be achieved “external” to the running program’s instruction stream.*

How can we activate the OS independently of the instruction stream?

- Interrupt signals, caused by events external to the processor, are sensed by the processor and causes a trap similar to what happens when decoding an ”illegal” instruction
- OS must make sure that interrupts happen. When they do, OS will be activated and may do whatever we want it to
- Interrupts are ensured by requesting recurring wake-up signals from a ”timer” external to the processor
- *Lots of finesse has to be added, but these are some of the basic mechanisms*

Sidebar: Interrups, traps, and exceptions

- Be aware of these terms as they are used inconsistently across the field of computing
- Ask yourself
  - Are we talking about events internal to the processor, generated synchronously with the instruction execution stream
  - Or is it an external event, generated asynchronously with the processor, and

Sidebar (cont.): Interrups, traps, and exceptions

- Ask further
  - Is occurence of the event to to be handled by the OS
  - …Or by user supplied code
- For numerical computations it may well be wise to let user programs specify how some possibly probelematic events are handled
  - *You may want to check William Kahans assault on Java for en entertaing and enlightening intro to some of the numerical issues.*
    - http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf

What Is an Operating System?

- Extension of hardware
- Virtual machine interface and physical machine interface
- Coordinate between applications and hardware resources (concurrency, device drivers, memory, file system and networking)
- Standard services (library, window system)
- What if you don't have an OS? source code -> compiler -> object code -> hardware
- What if you run an application at a time? Early DOS, Libraries, device drivers, interrupt handler, No overlapping

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*

Approaches to Teaching Operating Systems

- Paper
- Smaller exercises using existing operating systems
- Modifications to exisiting systems
  - Emulator
    - NachOS
  - "Metal", bare machine
    - Unix, Linux
    - Minix
- Roll your own

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 instead of SW, OS and computer vendors ☺

Our Approach

- 5-6 projects (some or all are mandatory)
  - From boot to a usefull OS kernel w/demand paging
  - 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 synchronized
- 3 weeks/project
- Design Review during first week of each project
- Linux, C, assembler
- Close to the computer, but VMware useful to reduce the number of reboots

Projects

- P1: Bootblock, createimage, boot first ”kernel”
- P2: Non-preemptive scheduling, simple syscalls, simple locks
- P3: Preemptive scheduling, syscalls, interrupts, timer, Mesa style monitor
- P4: P3 functionality+keyboard interrupt & driver, message passing, simple memory management, , user level shell
- *P5: P4 + demand paging memory management*
- P6: File System

The Competition

- At the end of the course
- Benchmark
  - Checks
    - OS functionality
    - OS performance

Platform

- PC with Intel Pentium or better
- Floppy drive
- **Linux Redhat**
- Language C (gcc) and assembler (gnu)
- PC emulatorer
  - VMware (should work)
  - Bochs (may work, to be determined)

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, (Lars A. Bongo fixes many bugs and adds a FAQ and more), 1999
  - Tromsø/Princeton/Oslo
    - D240/COS318/INF242, 2001, Vera Goebel, Thomas Plagemann, Otto Anshus

Literature

- *Modern Operating Systems*, by Andrew (Andy) Tanenbaum, Prentice-Hall, 2002
- 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

Topics

- **Protection & System Calls**
- **OS Structures, Process, Threads**
- **Non-preemptive Scheduling**
- **Threads & Mutex**
- **Preemptive Scheduling & Mutex**
- **Semaphores, Eventcounts, Monitors**
- **Thread packages**
- **CPU scheduling**
- **Deadlocks**
- **Message Passing**
- **I/O devices & Drivers**
- **Address Translation**
- **Paging**
- **VM Design Issues**
- **Disks & Files**
- **File systems & FS implementation**

60's vs. 00's

- Today is like in the late 60s. OS's are enormous
  - small OS: 100k lines
  - big OS: 10M lines
  - 100-1000 people years
- But ~90% is device drivers
- Project 5 ("post files"): ~6500 lines ☺

OS design tradeoffs change as technology changes

	1981	1998	Ratio
SpecInt (92)	1	800	1:800
$ / machine	$100k	$2000	50:1
$ / SpecInt	$100k	$4	25,000:1
DRAM	128k	128M	1:1000
Disk	10M	10G	1:1000
Network Bandwidth	3Mbits/sec	1.28Gbits/sec	1:400
Address bits	16-32	32-64	1:20
Users / machine	10s	1 (or <1)	> 10:1

HW is expensive, human cheap

- User at console, OS as subroutine library
- Batch monitor (no protection): load, run, print
- Data channels, interrupts: overlap I/O and CPU, **multiprogramming**
- DMA
- Memory protection: keep bugs to individual programs
- Multics: www.multicians.org OS designed 1963 and ran in 1969,
- Multics, the foundation of Unix www.multicians.org/general.html#tag14
- IBM 360 OS released with 1000 bugs

HW cheap, human expensive

- Use cheap terminals to let multiple users share a computer, **interactive timesharing**
- Unix enters the mainstream
- Problems: thrashing as the number of users increases

HW cheaper, human more expensive

- Altos OS: Ethernet, Bitmap display, laser printer and pop-menu window interface, e-mail, publishing software etc
- Personal computers
  - Initially library and later with memory protection and multiprogramming

Parallel and Distributed Systems

- Networked computing - Network is the computer
- Cluster of Multi Processors (CLUMPS)
- Cluster of PCs
- Network of Workstations (NOW)
- GRID of clusters
- Global clusters
- GRID