INF-3150/60 Operating Systems
Introduction

Fall 2005
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
- By end of semester you will be "King of the Hill"

People

- Vera Goebel, Ifi, UiO & UiTø
- Thomas Plageman, Ifi, UiO
- Otto J. Anshus, Ifi, UiT & UiO
- Others
  - Tore Larsen, Ifi, UiT?
- Teaching Assistents (TA's)

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?

Microsoft Windows xp

QNX

Mac OS

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. (Source Knut Skog, UiT)
- More by Lundh at:
  - http://www.ifi.uio.no/ansatte/yngvar.html

[Sidebar] Penetration of Microprocessors

Ant. distinkte prosessorer solgt i perioden 1998 -- 2002

	Server	Desktop	Embedded
1998	3,000,000	93,000,000	290,000,000
1999	3,000,000	114,000,000	488,000,000
2000	4,000,000	135,000,000	892,000,000
2001	4,000,000	129,000,000	862,000,000
2002	5,000,000	131,000,000	1,122,000,000

Data: Patterson & Hennessy: CODA, 3rd.Ed., 2004

The disparity in numbers was commented pointedly by Nick Tredennick abt. a decade ago.

Compiter Science and the Microprocessor, DDJ, June 1993

Cell-phones have OS'es?

- Symbian
- Linux
- Microsoft

Cameras have OS'es? Camera Operating System (COS)

- FlashPoint
  - Pentax, Olympus, Sanyo, Kodak
- Toshiba

Cameras also have software for exposure and focus

Cars have OS'es?

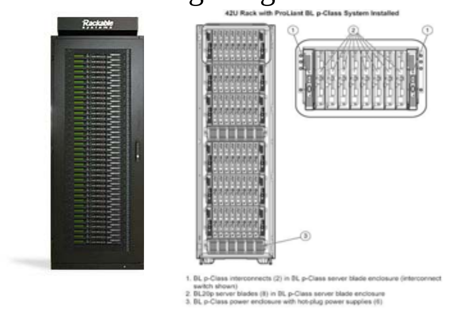
Magneti Marelli's engine control unit running Wind River's VxWorks software; it's used by Ferrari and other teams.

- http://www.autofieldguide.com/articles/100305.html
- 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 …And are getting smaller


Rackable
42U Rack with ProLiant BL p-Class System Installed
1
2
1
3
1. BL p-Class interconnects (2) in BL p-Class server blade enclosure (interconnect switch shown)
2. BL20p server blades (8) in BL p-Class server blade enclosure
3. BL p-Class power enclosure with hot-plug power supplies (6)

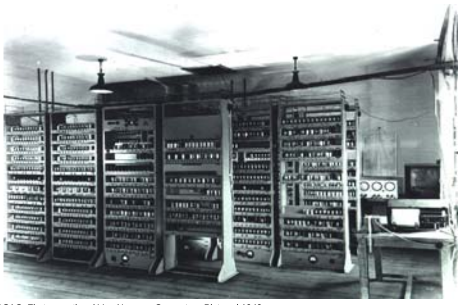
Observations and Questions

- 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 (Rochester, Accent, Mach) reportedly wanted to see Mach in light switches (which are currently being computerized)
  - Footprint: small vs. large

OS, other programs, and HW

- OS and other programs use the hardware (processor, memory, input/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

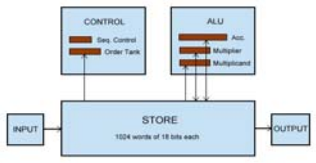
http://www.dcs.warwick.ac.uk/~edsac/

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 (János von Neumann, 1903-1957)
- First operational Von Neumann machine
- Stored program concept
- Also
  - Subroutines
  - Bootstrap (shadow ROM BIOS)
    - 31 instructions, second release (3 months later): 41 instructions
    - Get input, assemble in memory, start first instruction

EDSAC Architecture



```
repeat {
        fetch instruction;
        decode instruction;
        fetch operands;
        execute instruction using operands;
        write result somewhere;
}
```

- Implementation
  - 3000 vacuum tubes/valves
  - Memory: mercury delay lines
  - Input/output: paper tape/teleprinter
- Most *architectures* since have been like this

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

- We assume that the OS has been “booted”: power on/start up code reads boot block from HD to memory/boot block code reads OS from HD to memory/OS is given control
- 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, faults, 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.): Interrupts, traps, faults, and exceptions

- Ask further
  - Is occurence of the event 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 problematic 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?

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

- Software between applications and hardware
- Make finite resources “infinite”
- Provide protection and security

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

- 6 projects, all 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
- 2-3 weeks/project
- Design Review during first week of each project
- Linux, C, assembler
- Close to the computer,
  - but emulator (Vmware/Bochs/Virtual PC) can useful to reduce the number of reboots
  - Or have an extra PC to try your code on

Projects

- P1: Bootup
  - Bootblock, createimage, boot first "kernel"
- P2: Non-preemptive kernel
  - Non-preemptive scheduling, simple syscalls, simple locks
- P3: Preemptive kernel
  - Preemptive scheduling, syscalls, interrupts, timer, Mesa style monitor (practical version of the original Hoare monitor), semaphores (Dijkstra)
- P4: Interprocess communication and driver
  - P3 functionality+keyboard interrupt & driver, message passing, simple memory management, user level shell
- P5: Virtual memory
  - P4 + demand paging memory management
- P6: File system

The Competition

- At the end of the course
- P5
- Benchmark
  - Checks
    - OS functionality
    - OS performance
- Don't take it too seriously, it is meant to be fun
  - But nice prices…
  - Honour and fame…

Platform

- PC with Intel Pentium or better
- Floppy drive
- **Linux Redhat**
- Language C (gcc) and assembler (gas from gnu)
- PC emulatorer
  - VMware
  - Bochs
  - Virtual PC (?)

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), Peter Jensen (initialization), Sigurd Sjursen (help debugging interrupt software/hardware), 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 working (sort of) code 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 (Typical Systems Used at Universities)

	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**

Comparing a typical Academic Computer 1983 vs. 2004

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.

Computation & Communication

- Performance/Price doubles every 18 months:
  - *Exponential* Growth (!)
    - 100x per decade
    - Progress in next 18 months == ALL previous progress…
    - New storage == sum all old storage ever
    - New processing == sum of all old processing
    - Aggregate bandwidth doubles in 8 months

Phase 1 HW Expensive, Human Cheap

- Assumptions
  - No bad people, no bad programs
  - Minimum interactions
- User at console, OS as subroutine library
- Batch monitor (no protection): load, run, print
- Developments
  - Data channels, interrupts: overlap I/O and CPU, **multiprogramming**
  - DMA
  - Memory protection: keep bugs to individual programs
- Exceptions to the rule
  - 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

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.

Phase 2:
HW Cheap, Humans Expensive

- Use cheap terminals to let multiple users share a computer: **interactive timesharing, time-sharing OS**

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

- Unix enters the mainstream
- Problems: thrashing as the number of users increases

Phase 3
HW Cheaper, Humans More Expensive

- Personal Computer
  - Altos OS: Ethernet, Bitmap display, laser printer
  - Pop-up menu window interface, e-mail, publishing software, ftp, telnet, …
  - Eventually >100M units/year
- PC OS
  - Initially library and later with memory protection and multiprogramming
  - networking

Phase 4: >1 Computer per User

- Parallel and distributed systems
  - Parallel machine
  - Clusters
    - Uni-processors (single and multi-core)
    - Multi-processors (single and multi-core)
    - Grid
      - Multiple computers
      - Multiple clusters of computers
- Pervasive computers
  - Wearable computers
  - Computers everywhere, invisible
- OS both specialized and general

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Why OS is not Trivial

- Simple OS is inefficient
  - If process is waiting for something, machine sits wasted
- Obvious idea
  - Run more than one process "at once"
  - When one process blocks, switch to another
- Problems
  - What if a program runs an infinite loop
  - …or starts to randomly access memory
- OS should (must) add and have protection

Why OS is not Trivial

- Simple OS is expensive
  - One user == one computer
- Obvious idea
  - Allow N users "at once"
  - Does machine now run N times slower?
- Problems
  - What if users are evil
  - … or too many
- OS must add protection

Protection at 10,000 feet

- Protection is to isolate "bad" programs and users/people
  - Preemption, interposition, and privilege ops
- Preemption
  - Give apps and users something, but can take it away
- Interposition
  - OS between app and HW resources
  - Track/watch resources given to apps and users
  - On every access, check that access is legal
- Privileged/unprivileged mode
  - Apps and usersd are unprivileged
  - OS is master and commander

Successful Protection Examples

- Protection CPU by using **preemption**
  - Clock interrupt: HW periodically suspends (app), and give control to OS (or some special code at known places)
  - OS decides whether to take CPU away from app or not, and who to give it too
- Protecting memory by using **address translation**
  - Every **load** and **store** instruction checked for legality
  - Typically use this "machinery" to translate to another value

Real Systems have Holes

- Most protect some things, but ignore others
- Many will have trouble running

```
int main(){
        while(1)
                fork();
}
```

  - Common response: freeze
    - Reboot to unfreeze
  - Assume "stupid", but not malicious users
- Duality
  - Technical: have processor & memory quotas
  - Social: help users to learn more, yell at malicious users

Fixed Pie with Infinite Demand

- How to make "pie" go further?
  - Resource usage is *bursty*
    - Give to others when idle/waiting
- But, more utilization => more complexity
  - How to manage
    - One road/car vs. freeway
  - Abstractions (different lanes), synchronization (traffic lights), increase capacity (build more roads)
- But, more utilization => more contention
  - What to do when illusion breaks?
    - Refuse service (busy signal), give up (VM swapping), backoff and retry (Ethernet), break (freeway)

Finite -> "Infinite"

- Method 1: Exploit bursty apps
  - Take resources from idle guy (process, thread) and give to someone with something to do right now
- Method 2: Exploit skew
  - 80% of time taken by 20% of code
  - 10% of memory absorbs 90% of references
    - Cache 10% in fast memory, 90% in slow
- Method 3: Past predicts the future
  - Assume future == past
  - What's the best cache entry to replace?