Operating Systems and Networks

Lecture 01:
Introducing the course and
Introduction to OS-part 1
Behzad Bordbar
Introducing the course

- Why this module was created?
  Important
  First time
  Both practical and theory
  Requires lots of work and time... to prepare for exam you must start NOW!

But
- Does not require lots of background
Introducing the course (continue)

1. Demonstrate an understanding of the fundamental concepts and issues involved in OS and networking of IP-based systems. Interact and manage an Operating System.
2. Demonstrate an understanding of the challenges involved in the design of Distributed Systems in general and main methods of addressing them.
3. Explain Transport Layer protocols and their differences.
4. Understand the basics and practical issues and architectures involving in important Application Layer protocols.
5. Demonstrate practical understanding of the theoretical foundations of Operating Systems and Distributed Systems.
Introducing the course (continue)

- 1.5 hr examination (80%), continuous assessment (20%).
- Supplementary (where allowed): 1.5 hr examination only (100%).
- You will have lab hours to ask your questions from a teaching assistant (time will be announced later).
What am I going to learn?

Mixture of theory AND practice of:
• OS Fundamentals and architecture
• Unix shell
• Processes and threads
• OS networking
• OS support for Distributed Systems
• Distributed object, RMI and RPC
• Time
• You will also learn about as we go through the course
  • Virtualisation, Xen, KVM *
  • cryptographic algorithms and Security
  • P2P (*)
  • Wireless protocols
• But we don’t teach kernel programming
Opportunity to learn Unix

- I will use linux to teach OS concepts
- How do I start?
  - SoCS machines
  - Dual boots system
  - Virtualised environment (install Ubuntu on VMware player or Virtual box)
- Mac people are OK, as have access to shell.
Learn Basic Linux commands, for example from http://www.debianhelp.co.uk/commands.htm

Who wants a linux demo? Hands up!
What reading material can help me?

Course is based on chapters from

- **Operating System Concepts** by Abraham Silberschatz, Peter B. Galvin and Greg Gagne
- **Distributed Systems: Concepts and Design** by George Coulouris, Jean Dollimore, Tim Kindberg and Gordon Blair

We have excellent **library with lots of books** on OS, Networking and Distributed systems including multiple copies of the above. Among others:

- **Modern Operating Systems** by Andrew S. Tanenbaum
- **Data Communications and Networking** by Behrouz Forouzan

There are lots of books, papers, tutorial.... **online** too.

**How do I study this module and learn it!**
Opportunity to learn Unix

- I will use linux to teach OS concepts
- How do I start?
  - SoCS machines
  - Dual boots system
  - Virtualised environment (install Ubuntu on Vmware player or Virtual box)
  - Mac people are OK, as have access to shell.
- Learn Basic Linux commands, for example from http://www.debianhelp.co.uk/commands.htm
- Who wants a linux demo? Hands up!
We want to learn these? But what is OS?
central processing unit (CPU)

- “brain” of the computer
Semiconductor companies:

- **Intel**
  - PC: core i7, i5,....
  - Sever and Workstation processor: Intel Xeon*  processor E7, E5, ...
- Mac: Hawell, Ivy Bridge, Sandy Bridge,...

**AMD (Advanced Micro Devices Ltd)**
- Boldddozer (Vishera, Zambezi,...), K10 series (Athlon, Operon, ...)

All above are based on x86: **backward compatible (??)** set architecture based on Intel 8086 cpu.

AMD also produces CPU based on a blueprint developed by ARM
- Majority of mobile phones use ARM
How does it work?

- fetch the first instruction from memory,
- decode it to determine its type and operands
- execute it, and
- then fetch, decode, and execute subsequent instructions.

Instructions are CPU specific

- i7* core cannot execute ARM instructions!
Accessing memory to get an instruction or data takes much longer than executing an instruction:
- CPUs contain **registers** inside to hold variables and temporary results.

What is instruction set?
- **Load/store** data from/to memory into a register
- **combine** two operands from registers and store result, for example adding
- ...

CPU(continue)
Register

- 8-bit or 32-bit storage. Examples of registers are:
  - **General purpose** reg.: temporary data & results
  - **Program counter**: memory address of the next instruction to be fetched and then program counter is updated to point to its successor.
  - **Stack pointer**: points to the top of the current stack in memory

**Stack?**

- Stack contains one frame for each procedure to be entered
- Stack frame holds those input parameters, local variables, and temporary variables that are not kept in registers.
CPU modes

- At least two modes, **kernel mode** and **user mode**
  - **Kernel mode**: CPU can execute every instruction in its instruction set and use every feature of the hardware (complete access to hardware).
  - When CPU in kernel mode we say OS in kernel mode.
  - User programs run in user mode, which permits only a subset of the instructions to be executed and a subset of the features to be accessed. Generally, all instructions involving I/O and memory protection are disallowed in user mode.
  - Program Status Word (PSW) is a register that (among other things) keeps the mode of the CPU.
System call and Trap

So how do a user program do for example I/O?
- Ask OS: a user program must make a system call, which traps into the kernel and invokes the operating system.
- The TRAP instruction switches from user mode to kernel mode and starts the operating system.
- When the work has been completed, control is returned to the user program at the instruction following the system call.

We will look at this in details later! Just one small point
Not every trap is caused by system calls!

- Some traps are caused by the hardware to warn of an exceptional situation such as an attempt to divide by 0 or an arithmetic underflow.
- Trap causes operating system gets control and must decide what to do. Example:
  - OS terminates program when error
  - error can be ignored and underflowed number set to 0
- Do you know about exception handling: that is when control is handled to program.
GPU ≠ CPU

- Graphic Processing Unit (GPU) coined by Nvidia
- Called VPU (Visual Processing Unit) by ATI (a competitor of Nvidia)
- GPU: electronic circuit specialised for graphic processing
- Presented as a video card in a PC for processing graphics
- Originally designed and used for image manipulation. Turned out to be suitable for parallel computing (most notably CUDA)
How to say what CPU you have

$ cat /proc/cpuinfo
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 58
model name : Intel(R) Core(TM) i7-3667U CPU @ 2.00GHz
stepping ....

You see a number i7-3667U, what does it mean?
See intel page
or run hardware lister

$ sudo lshw
$ sudo lshw |grep -i cpu