Sistemas Embebidos de Internet de las Cosas

2/4

Diego Méndez Chaves, Ph.D

Programación de Formación en Internet de las Cosas Intel-Javeriana
Noviembre 2015

MINTIC COLCIENCIAS Ciencia, Tecnología e Innovación

Pontificia Universidad JAVERIANA Bogotá

Agenda

- **The Internet of Things**
  - *Definitions and some stats*
  - *A simplified model*
- **The Embedded Platforms**
  - *Microcontrollers vs Processors*
  - *Do we need an OS?*
  - *The Intel Galileo Platform*
- **Why embedded linux?**
- **Some basic knowledge on Operating Systems**
- **The Linux filesystem**
- **The Integrated Decelopment Environments for Intel Galileo boards**
- **The Intel XDK IoT Edition**

The Lecturers

- **Antonio F. Mondragón Torres, Ph.D**

  is a Research Scientist at Intel Corporation, with extensive experience in research and development from design concept through development to implementation. He was an Assistant Professor at the Rochester Institute of Technology (RIT), Rochester NY, USA. Proficient in using critical design tools and languages for system, digital, analog and mixed mode design and verification. Highly effective in working with cross-functional/international teams. Antonio's research interest include Embedded Systems, Electronics.,FPGA, ASIC, VLSI and Communications. He holds a Ph.D in Electrical Engineering from the Texas A&M University, a M.Sc in Electrical Engineering from the Universidad Nacional Autónoma de México, and a B.Sc in Electronics and Communications from the Universidad Iberoamericana.

- **Diego Méndez Chaves, Ph.D**

  is an Assistant Professor in the Department of Electronics Engineering at the Pontificia Universidad Javeriana, Bogotá, Colombia. He received his Ph.D (2012) and his M.Sc. (2011) in Computer Science from the University of South Florida, Tampa FL, USA, his M.E. (2008) from the Universidad de Los Andes, Bogotá, Colombia, and his B.E (2005) in Electronics Engineering from the Universidad Nacional, Bogotá, Colombia. Diego's research interests include participatory sensing, digital systems design, operating systems (RTOS, embedded Linux, etc.), high-level systems design, parallel architectures, wireless sensor networks, and co-design techniques.

Principios en IoT

- **Dates:** April 2016
- **Intensity:** 32 hours (4 sessions of 8 hours each)
- **General content:**
  - Introduction to IoT.
  - Applications and Business Trends in IoT.
  - Overview of standard sensors (accelerometer, temperature, etc).
  - Review Mini Sensors project submissions.
  - Introduction to Physical Computing.
  - Introduction to the Galileo Micro-controller.

Sistemas Embebidos de IoT

- **Dates:** November 6 – 28, 2015
- **Intensity:** 40 hours (4 sessions of 10 hours each)
- **General content:**
  - Overview of Embedded Systems.
  - Embedded Platforms (Quark SOC, Intel Galileo).
  - Basic Concepts on Operating Systems
  - Intel XDK IoT Edition Programming
  - Threads and process synchronization
  - An overview on real-time scheduling and multicore embedded systems overview.
  - Project formulation and execution.

Redes de IoT

- **Dates:** January 22 – February 13, 2016
- **Intensity:** 40 hours (4 sessions of 10 hours each)
- **General content:**
  - Introduction to Computer Networks.
  - Networks for IoT (Wireless Sensor Networks, Enabling technologies, Network protocols for IoT).
  - An IoT-based Sensing Platform.
  - Introduction to Wind River Intelligent Device Platform XT.
  - Hands-on lab Wind River gateways.

Sensores de IoT

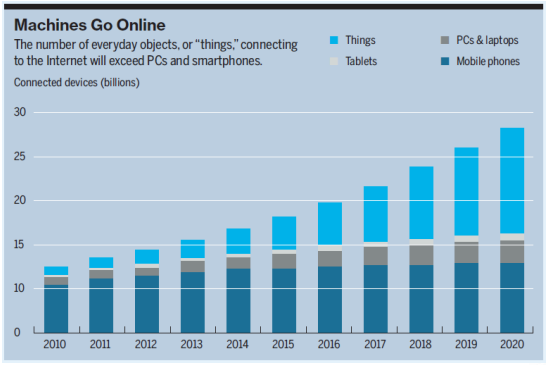
- **Dates:** February 19 – March 12, 2016
- **Intensity:** 32 hours (4 sessions of 8 hours each)
- **General content:**
  - Introduction of Sensors and emerging trends.
  - Bio-Sensors and Data Analysis.
  - Health and Fitness sensors overview and accelerometers.
  - GSR and temperature sensors.
  - Machine vision sensing techniques.
  - Emerging sensor fabrication and micro-sensing

The fundamentals

THE INTERNET OF THINGS

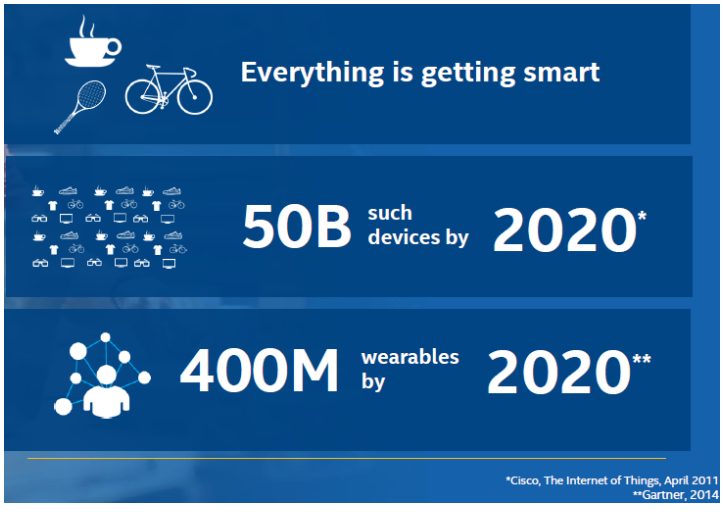
The Internet of Things

- Population of the world: 7.2 Billion
- Population of Internet users: 3 Billion
- By 2020, expected connected devices:
  - 26Billion by Gartner
  - 30Billion by ABI
  - 50Billion by Cisco



2014 study

Humans & Technology

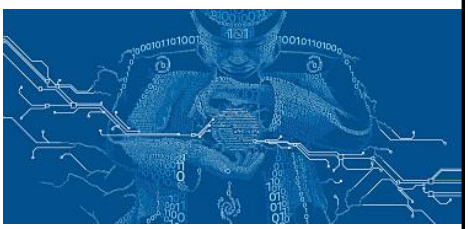


Where did it all begin?

- The **origin** of the term 'IoT'
  - Coined in 1999 by the founders of **MIT Auto-ID Center,** in the **field of identification technologies** (**RFID)**
  - associated with an **open infrastructure** allowing computers to automatically **identify man-made objects and track them** while flowing from the plant to the distribution center
- The **evolution** of the term 'IoT' and the current meaning
  - An **heterogeneous** network connecting computers to objects
    - **affordable hardware** (not just RFID tags and readers)
    - network **software and protocols**
    - **languages** for **describing objects** in ways computers can understand

IoT Definition

The **Internet of Things (IoT)** is the **interconnection** of **uniquely identifiable embedded computing devices** within the **existing Internet** infrastructure. (*Wikipedia*)



- '***Interconnection***' refers to networking (mostly wireless)
- '***Uniquely identifiable***' reminds addressing (IPv6)
- '***Embedded Computing***' reminds of processing capabilities, reduced size and full integration of components
- '***Existing Internet***' reminds IoT as the next evolution of the current Internet

The essential: the IoT is seen as the Internet of the Future, characterized by a very large population of objects

  - It says much but not all!

Internet of Things

Many definitions, but main characteristics of IoT are :

- **Highly connected**
- **Smart**
- **Thing** = Not necessarily a computer/phone/tablet
- Network/physical world **interface**

Highly connected

Can be internet of course, but can also be :

- **Intermittent** internet (to save power, or because it's unavailable)
- Low bandwidth, long distance (**Sigfox**)
- **Mesh** networks
- Local: **BLE, ZigBee**

The "Internet" of IoT is not your usual Internet

Smart

Smart does not mean the raw information is smart or coming from an ultra precise sensor. Smart if :

- You **cross analyze** data from basic sensors
- You perform a first level of **AI locally**
- You improve your solution with a **central AI**
- It works with an **imperfect connectivity**

Don't wait for the perfect sensor.

Thing

- A **laptop** will use **90%** of your brain. It's designed as an exclusive tool.
- A modern **touch phone** will use **40%**
  - You can walk, but not drive safely.
- An **old phone** with keys will use **20%**

IoT solutions : target 0-5% maximum

Thing

Why 0-5% maximum ?

- You may have 50 of them at home,
  - You don't have time for each of them!
- You don't want to think while you interact with them. Natural interfaces are better.
- The best interaction: no active interaction.

No screen, no settings, limited features.

Physical world / Network

An embedded project may not be IoT if :

- no information is gathered from the environment by analog or digital **sensors**
- no physical action is taken by **motors**, lights, sound, …

Plan to interact with the physical world

Why deploy IoT solutions ?

- **To optimize :** an automated air conditioning system could improve your daily life.
  - But it can also allow the electricity company to limit your power consumption during summer peak days.
- **20% cheaper electricity** all year to allow the electricity company to limit by 10% your consumption during 3 peak days a year ?

… why not :-)

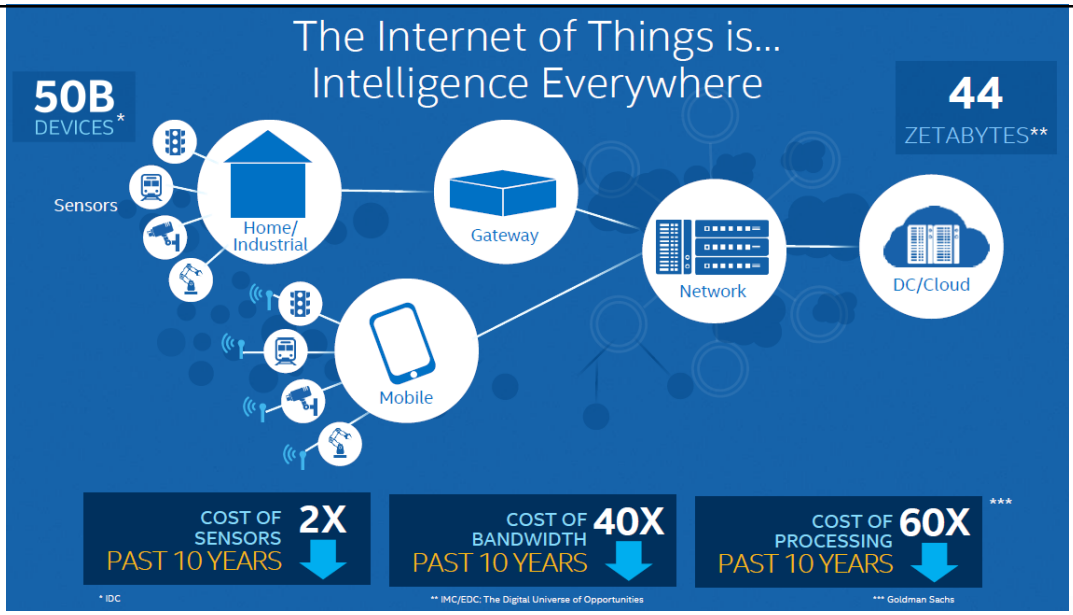
Think hardware + software data service rather than just hardware.

Why deploy IoT solutions ?

- **To collect data :** Internet access and communications are already highly monitored to characterize your behavior.
  - But the physical world is not, except for GPS.
  - Knowing everything about your electrical devices, consumptions, movements at home has a lot of value.
  - Happy ? …

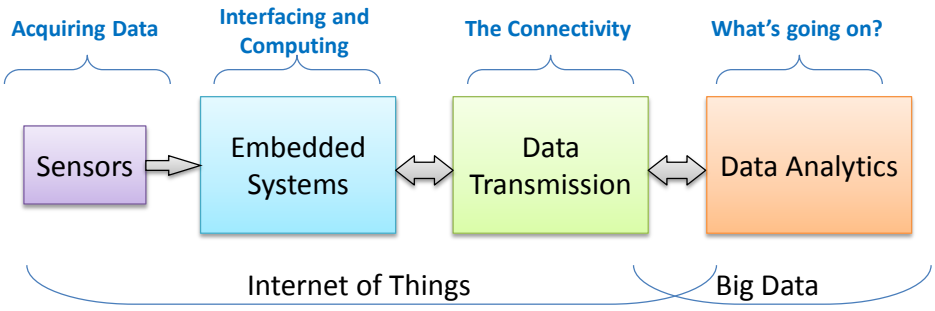
… maybe not :-(

Just like with mobile devices, a lot of IoT devices will collect data.



A Basic IoT Model

- A very simplistic model for the IoT would consider the following modules:



An Intel IoT Model

Interfacing and Computing

THE EMBEDDED PLATFORMS

Microcontroller vs Processor

- If you use Arduino you are using a 8-bit **microcontroller**. It's simple and predictable for people new to software development.
  - But it's impossible to install a full system like linux, so it's limited by nature.
- To have a full OS with network, the programming language of your choice and a lot of potential, a **processor** is required.

Digital / Analog IO

- If you develop on your laptop, you can only plug devices with high level IO like USB. But most IoT sensors are a lot simpler than that, using low level analog and digital IO.
- To be really useful as an IoT platform, **you must have digital and/or analog IO.**
- Note : Some IoT platforms only have digital IO, others like Intel Galileo and Intel Edison have both digital and analog IO.

Graphics

- Many wearable projects like watches have a touch screen, or a projector for glasses. They look a lot like small mobile devices, and require graphics.
- We have such platforms at Intel.
- But for all the other IoT projects, you don't need and don't want graphics.

Remember: display = interaction
= brain attention share = not scalable.

Networking

- **Network connectivity is important in IoT.**
- You may take a platform without wireless internet and add a **dongle** for prototyping, but :
  - It's harder to switch to production
  - Power optimizations are limited
  - Dongles often propose **limited features**: Bluetooth instead of Bluetooth Low Energy.
  - Integration with software-OS is not always trivial.
- Advice: **take platforms with great networking inside** for prototyping then production.

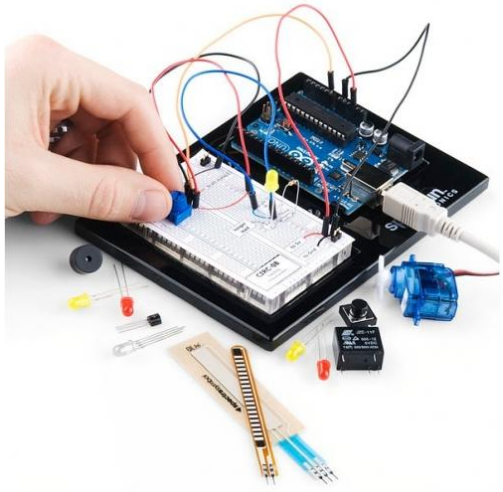
Form factor

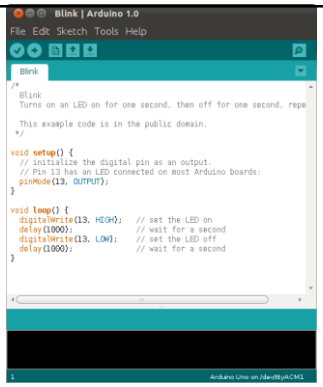
- It's convenient to have **large boards with lots of IO for prototyping**. But you'll need to design a **new board for production**.
  - It takes a lot of money, time and skills.
- Or you take a board with a **modular design**: your compute module has all the complex parts you won't redesign, and the simpler connectivity board can be replaced or designed easily.

Power features

- A big difference between IoT prototypes and production is **total power consumption**.
- You would not plug a 3G dongle on a desktop and call it a mobile phone, right ? Same for IoT.
- You need :
  - a very **efficient processor**, with advanced sleep/hibernation features
  - **power optimized wireless**
  - great **integration** of all parts
  - lots of software, driver and **OS optimization**

Arduino





ARM-based Microcontrollers platforms

Embedded Boards on the Market


pcDuino

Arduino Yun

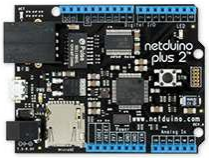

zedBoard


ifc6410


BeagleBone Black


Odroid-u3

gumstix



NetDuino


RaspberryPi

Galileo

Programs for Microcontrollers and Processors

- Programs for *any* processors fall into a few different classes: **firmware**, **bootloaders**, **basic input-output** systems, and **operating systems**.
- BTW**, even the most powerful server** or multimedia processor has to have **a piece of firmware** put on it with a hardware programmer at first.

Firmware	Stored on	Details
Single program	Processor's program memory	Is the only program running; must be loaded by hardware programmer
Bootloader	Processor's program memory	Must be loaded by hardware programmer; Takes small amount of program memory; can load another program into the rest of program memory
BIOS ***Basic Input-Output System***	Processor's program memory	Usually loaded by bootloader; can load operating system into RAM memory

Programs for Microcontrollers and Processors

- **Generally**, the term ***microcontroller*** **refers to firmware-only processor**, and a processor that runs an **operating system** from external storage is called an **embedded processor**, or a central processor
- For example, the **Arduino is a microcontroller**. **The Raspberry Pi, the BeagleBone Black and the Intel Galileo are embedded processors**, and phones, tablets, and laptops are multi-processor devices running a central processor.

Software	Stored on	Details
Operating System	External mass storage	Runs other programs; loaded into RAM by BIOS; unloaded from RAM on reset
Applications	External mass storage	Loaded into RAM by operating system and unloaded as needed

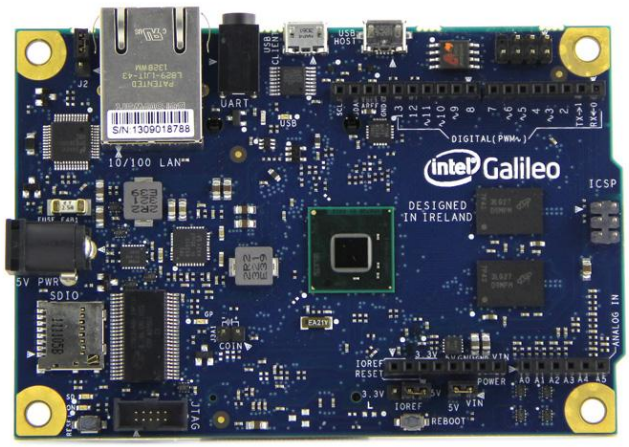
OS

- We all like to prototype with our desktop OS.
- It can be a big linux distro like **Ubuntu**, **Windows 10**, **OSX** … **it's easy**, all the packages are readily available.
- But a **professional grade embedded project** requires to start from scratch, **control each piece of code** added to the system and integrate with a large team of software developers.
- A typical open source OS for professional projects is **Yocto**.

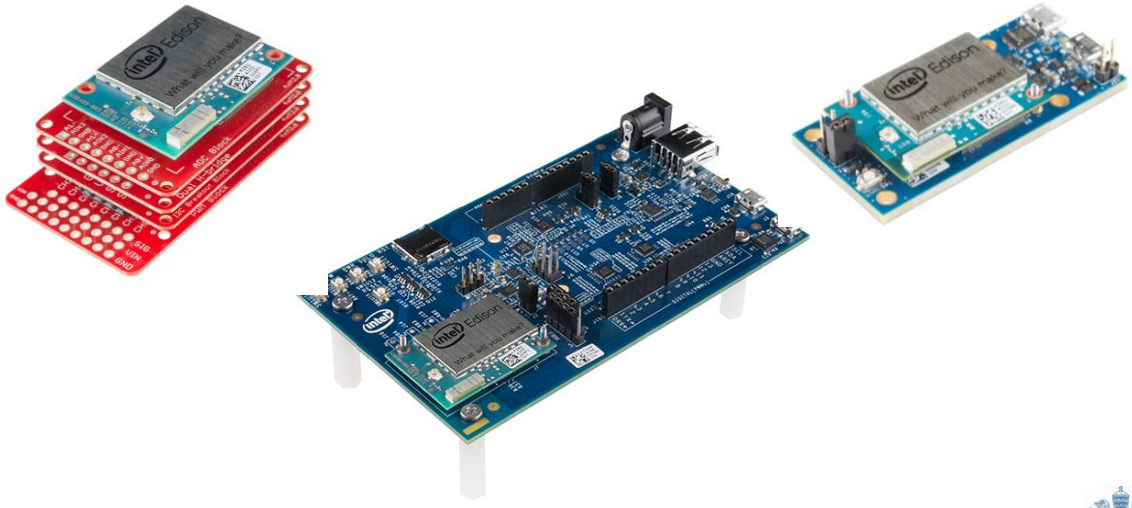
Toolchains and Development Environments

- The two most common languages for microcontrollers are:
  - **Assembly language,**
  - or **C language.**
- A **toolchain** is the combination of **compilers and linkers** needed to convert the instructions you write into a binary file that the **microcontroller** can interpret as its **instructions**.
- **Every manufacturer** and processor family has **its own assembly language**, but there's a **C compiler** for almost **every microcontroller** (mostly open):
  - ***avr-gcc***
  - ***msp430-gcc***
  - ***arm-gcc***
- An **integrated development environment (IDE)** helps the **programmer** to connect to the **toolchain**. An IDE usually contains a text editor with user interface elements to send your text to the toolchain and upload the result to the processor.

Intel Galileo



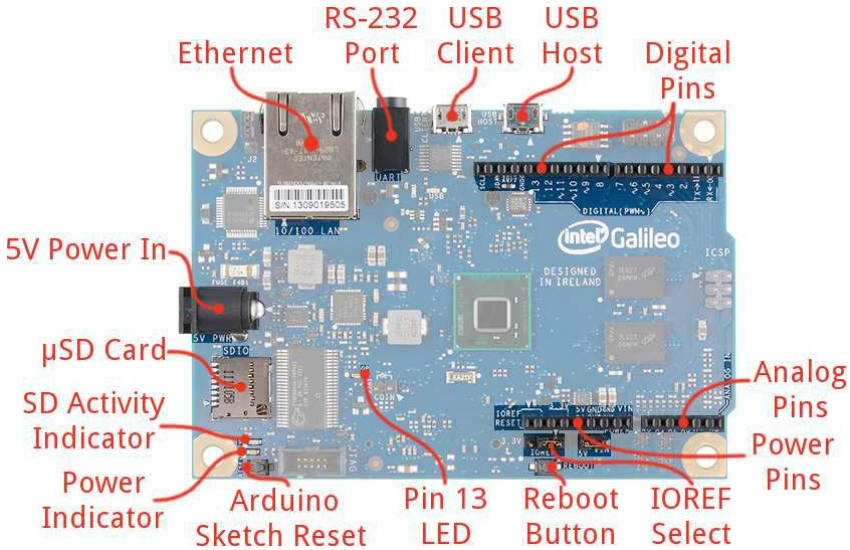
Intel Edison



Intel Curie

INTEL GALILEO PLATFORM

Intel Galileo Gen 1



Galileo General Features

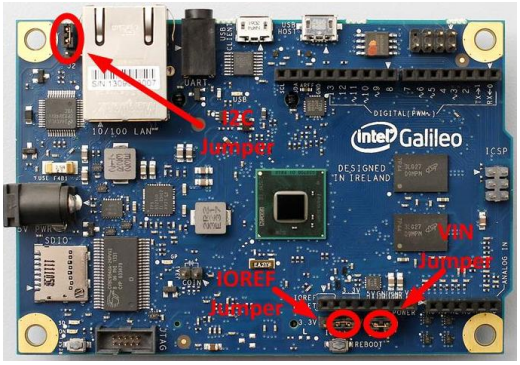
- **Quark X1000 SoC** @ 400MHz
  - 32-bit ISA architecture, 16 KBytes of L1 chache, 512 KByte embedded SRAM
- **RTC** (3V button battery optional)
- **Ethernet** 10/100 Mb
- **PCI Express mini-card** in the PCIe 2.0 standard (designed to connect a WiFi)
- **USB Host 2.0** (up to 128 devices)
- **USB client** for downloading the sketch, and connecting USB 2.0-compatible devices
- **Standard JTAG** connector
- **Two buttons** (reboot the processor, and reset the Arduino sketch)
- **Mass Storage:**
  - 8 MB SPI Flash memory (GNU/Linux bootloader and the last sketch loaded)
  - 512-Kbyte SRAM and 256 MB DRAM memory managed directly by the OS
- **11-Kbyte EEPROM** memory that can be programmed from the sketchs
- **Micro SDCard** up to 32GByte capacity → Yocto/Debian distribution
- **3.5mm jack connector** for a second serial port standard UART (not an audio input/output)
  - This idea was not successful in later versions

Galileo General Features

- ***Arduino compatible connector:***
  - **14 I/O pins** reproducing Arduino's configuration, six of which can be PWM. Each pin support both 5 and 3.3 V levels and a current of 10mA output and 25mA input.
  - **Analog input pins**, A0 to A5, are connected to a AD7298 integrated.
  - **I2C (TWI)** and can be interfaced with the Wire library;
  - **SPI bus** configured to 4MHz for compatibility with Arduino's boards but may be increased to 25MHz. Galileo board can operate only as a master.
  - **UART** port
  - **ICSP 6-pin connector** for in-circuit programming, to be compatible with the existing shields;
  - **5V and 3.3V outputs** provide a maximum current of 800mA each
  - AREF is the voltage reference for the ADC convertors. You cannot use this pin on the Galileo board.

Galileo Jumper Configuration

- **IOREF Jumper**
  - To support both 3.3V and 5V shields, the external operating voltage is controlled via a jumper.
  - The input range of the Analog pins is controlled by the IOREF jumper and must not exceed the chosen operating voltage. However, the resolution of AnalogRead() remains at 5 V/1024 units regardless of IOREF jumper setting.
- **I2C Address Jumper**
  - To prevent a clash between the I2C Slave address of the on board I/O expander and EEPROM with any external I2C Slave devices, jumper J2 is used to vary these addresses.
  - With J2 on pin 1 (white triangle), the 7-bit addreses are I/O Expander 0100001 and EEPROM 1010001. The other position defines the address to 0100000 and 1010000, respectively.
- **VIN Jumper**
  - The VIN pin can be used to supply 5V from the regulated power supply to attached shields or devices.
  - If there is a need to supply more than 5V to a shield, then the VIN jumper should be removed to break the connection between the on-board 5V supply and the VIN connection on the board header.



Intel Galileo SW

The board can run :

- **Linux Yocto** by default from Intel.
  (Yocto is an open source Linux used by embedded professionals)
- **Debian** variant by the community.
- **Arduino** style code by an emulator.
- **Windows** by Microsoft.
- In some cases, **WindRiver** solutions.

A brief Comparison

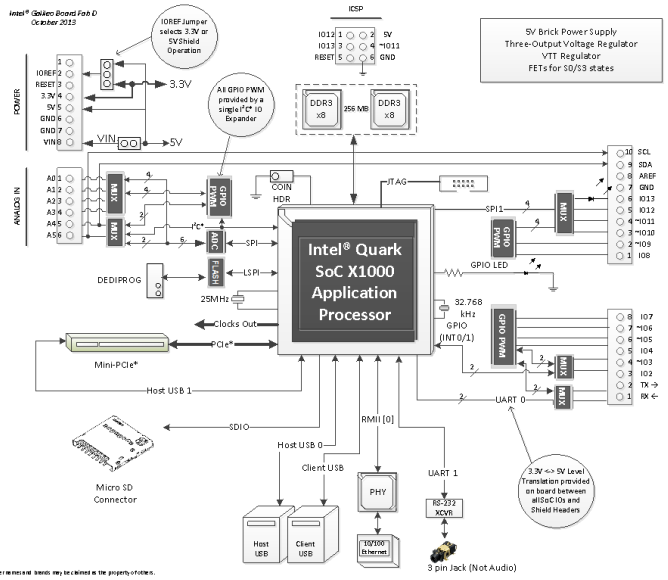


Feature	Beaglebone Black	Intel Galileo	Raspberry Pi
SoC	Texas Instruments AM3358	Intel Quark X1000	Broadcom BCM2835
CPU	ARM Cortex-A8	Intel X1000	ARM1176
Architecture	ARMv7	i586	ARMv6
Speed	1ghz	400mhz	700mhz
Memory	512MB	256MB	256MB (model A) or 512MB (model B)
FPU	Hardware	Hardware	Hardware
GPU	PowerVR SGX530	None	Broadcom VideoCore IV
Internal Storage	2GB (rev B) or 4GB (rev C)	8MB	None
External Storage	MicroSD	MicroSD	SD card
Networking	10/100Mbit ethernet	10/100Mbit ethernet	None (model A) or 10/100Mbit ethernet (model B)
Power Source	5V from USB mini B connector, 2.1mm jack, or header pin.	5V from 2.1mm jack, or header pin.	5V from USB micro B connector, or header pin.
Dimensions	3.4in x 2.1in (86.4mm x 53.3mm)	4.2in x 2.8in (106.7mm x 71.1mm)	3.4in x 2.2in (85.6mm x 56mm)
Weight	1.4oz (40g)	1.8oz (50g)	1.6oz (45g)
Approximate Price	$55 (rev C), $45 (rev B)	$80	$25 (model A), $35 (model B)
Documentation	Complete and open source.	Complete and open source.	Complete and open source.
Arduino compatible	No	Yes	No
Others	Real-time support, I2S audio, CAN.	Mini-PCI express slot, RTC	

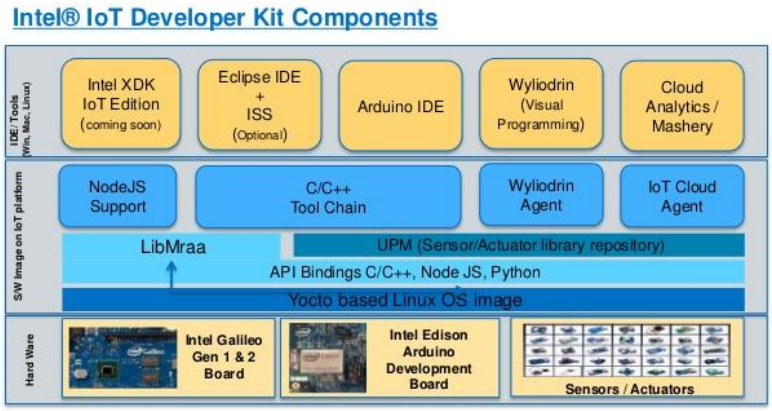
Galileo Pros!

- **Shields:** Compatible with most Arduino shields.
- **Familiar Arduino IDE:** The modified IDE also is capable of upgrading the firmware on the board.
- **Ethernet:** You can use the Arduino's Ethernet library.
- **RTC:** Most Linux boards rely on internet to get the current date and time. Use the on-board RTC and just wire up a 3.0V coin cell battery.
- **PCIe slot:** Connect WiFi, Bluetooth, GSM cards for connectivity, or even a solid state drive for more storage.
- **USB Host Port:** Use the the Arduino USB Host library to act as a keyboard or mouse for other computers.
- **MicroSD:** Use it through the standard Arduino SD card library.
- **TWI/I2C, SPI:** Use the standard Arduino Wire and SPI libraries.
- **Serial Connectivity:** There is a serial port to download the sketches, and there's another serial port for connecting to the Linux command line.
- **Linux:** A very light distribution of Linux is loaded onto the 8 MB of flash memory. If you want more sophisticated packages, you can boot a Yocto/Debian distribution from the SD card.

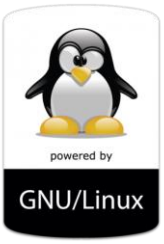
Hardware Architecture



Software Architecture



The SeeedStudio Grove Starter Kit for Intel Galileo



Tux

WHY EMBEDDED LINUX?

Advantages of Embedded Linux

- **Vendor and price independence**
- Advanced **network capabilities**
- Available Linux experts
- **A lot of hardware supported by GNU/Linux**
- You can **customize** Linux (drivers)
- Available **source code** (also documented)
- **Free support** through mail-lists, blogs, wiki, etc. (you can contact the code author)
- Good commercial support

Advantages of Embedded Linux

- Several **software applications** available:
  - Apache, boa, dhcp, asterisk, samba, …
- Different flavors:
  - Debian, Buildroot, Yocto, OpenEmbedded, OpenWRT, UbuntuCore, …
- **Interconnected world**:
  - IP, TCP, UDP, IPv4, IPv6, NFS, PPP, SNMP, HTTP, FTP, SSH, SAMBA, SMTP, POP, …
  - Bluetooth, Ethernet, WiFi, WiMax, …

Advantages of Embedded Linux

- And memory:
  - **Filesystems**: EXT2, EXT3, EXT4, FAT, YAFSS, JFFS, ReiserFS, NTFS, …
  - NAND memories, SD cards, MMC, Compact Flash, …
- And **hardware**:
  - USB, Serial, SPI, Ethernet, UART, …

And it is FUN!

Some basic concepts to get us started

OPERATING SYSTEMS

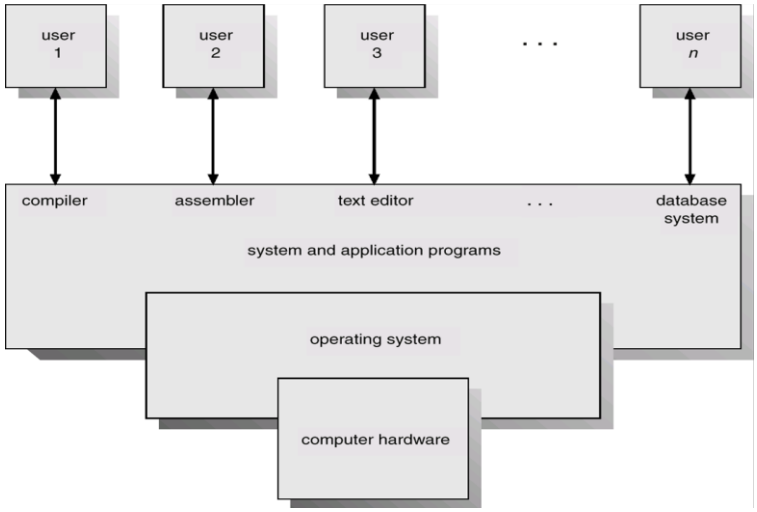
An Operating System?

- Two fundamental abstractions: **process** (defines the state of an executing program) and **operating system** (provides the mechanism for switching between the processes, among other things)
- **What is an Operating System?**
  - A program that acts as an intermediary between a user of a computer and the computer hardware.
- **What is the purpose of an operating system?**
  - To provide an environment in which a user can execute programs.
- **What are the goals of an Operating System?**
  - The primary goal of an Operating System is to make the computer system **convenient** to use.
  - The secondary goal is to make the computer system **efficient** to use.

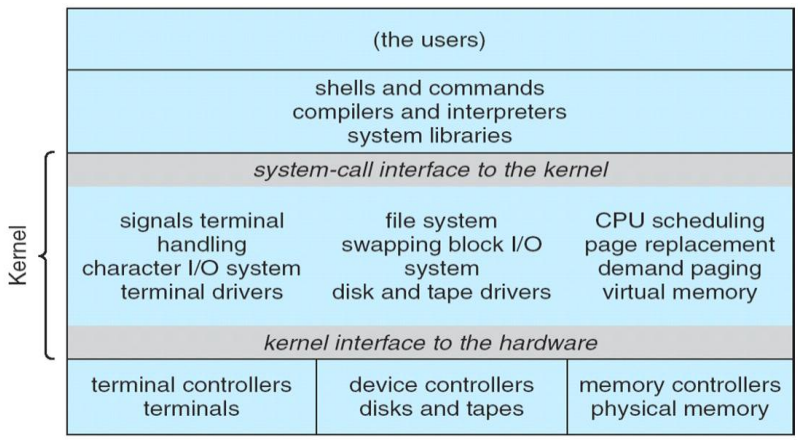
Computer System Components

- **Hardware** – provides basic computing resources (CPU, memory, I/O devices).
- **Operating system** – controls and coordinates the use of the hardware among the various application programs for the various users.
- **Applications programs** – define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).
- **Users** (people, machines, other computers).

Abstract View of System Components



Traditional UNIX System Structure



Function of Operating System

- **OS as Extended machine**
  - Computer Architecture shows that computer is made up of chips and wires
  - **We do not want to program on the bare metal**
  - Virtual machine creates a **hardware abstraction**
  - Abstract machine can **provide hardware independent interfaces**
  - Increase **portability**
  - Allow **greater protection**
  - Implication is that it is **much faster and easier to program** with less errors

Function of Operating System

- **OS as resource manager**
  - **Coordination and control** of limited resources such as memory, disk, network, etc
  - Deal with resource **conflicts**
  - Deal with resource **fairness**
  - Make **access efficient** as possible

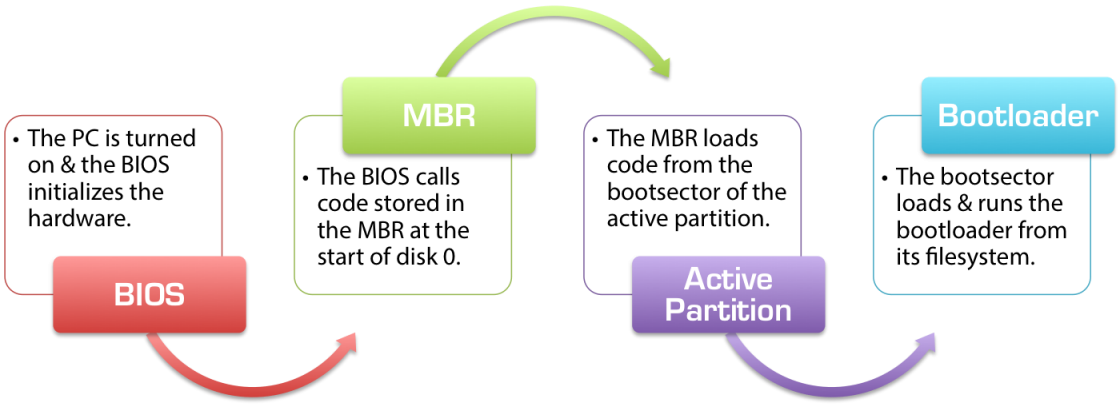
Parts of an Operating System

- **No universal agreement on the topic, but most likely**
  - Memory Management
  - IO Management
  - CPU Scheduling
  - IPC (inter-process communication)
  - MultiTasking/Multiprogramming
  - (On some Operating System, all this functionality is provided by a single program known as the ***kernel)***
- **What about?**
  - File System
  - Multimedia Support
  - User Interface (X, MSWin)

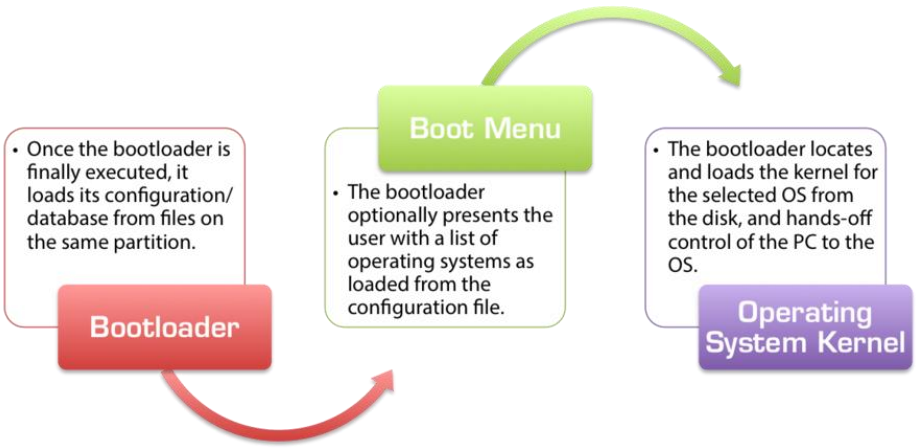
Evolution (History) of OS

- **First Generation:** Punched cards
- **Second Generation:** Transistors and batch systems.
- **Third Generation:** Integrated Circuits and important concepts:
  - ***Spooling***
  - ***Multiprogramming***
  - ***Multitasking***
  - ***Virtual Memory***

Booting Process



Booting Process



Monoprogramming

Memory

Spooling

- Stands for **Simultaneous Peripheral Operation On-Line**
- Allows for overlap of IO from one job with the computation of another job
- E.g., print spooling, which places a task into a queue for later processing.
- For instance, **while executing current job:**
  - **Read next job** from disk to memory
  - **Print previous job** to printer
- Disk is relegated to the role of a partitioned buffer

Job pool

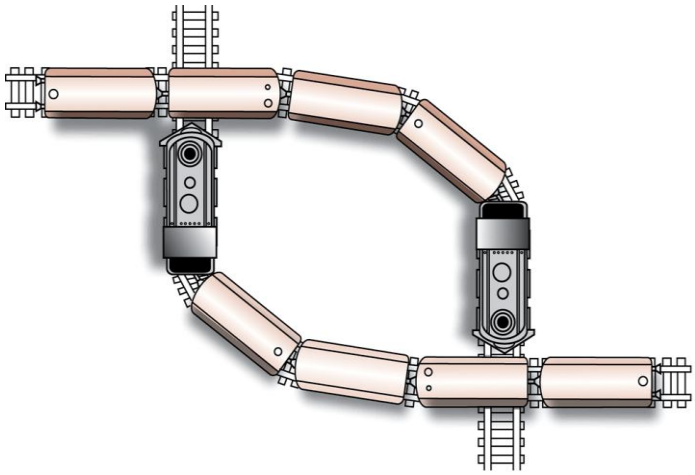
- **Advent of disk allows for random access**
  - Tape and card are sequential
  - Disks are sequential, but not as much :)
- **Several jobs can be waiting on the disk**
- The **job pool** is a data structure that contains info and points to the **jobs on the disk**
- We can now have job scheduling to determine the order in which the jobs run so that **CPU utilization can increase**.

Multiprogramming

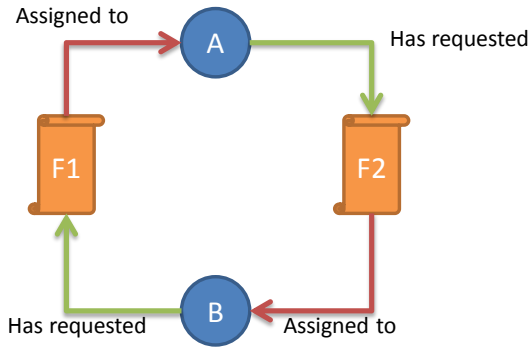
- Memory partitioned into several pieces
- CPU Starts a job
- **If the job is waiting for IO**, the CPU can **switch to another task**
- **Problems when having multiple programs?**
  - How to deal with shared resources?

Memory

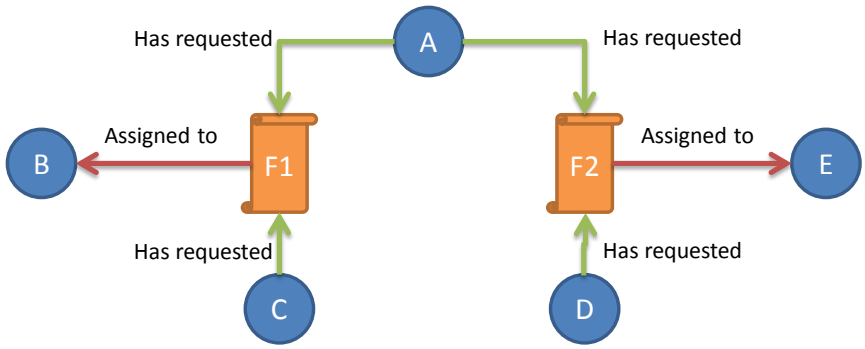
Deadlock



Deadlock

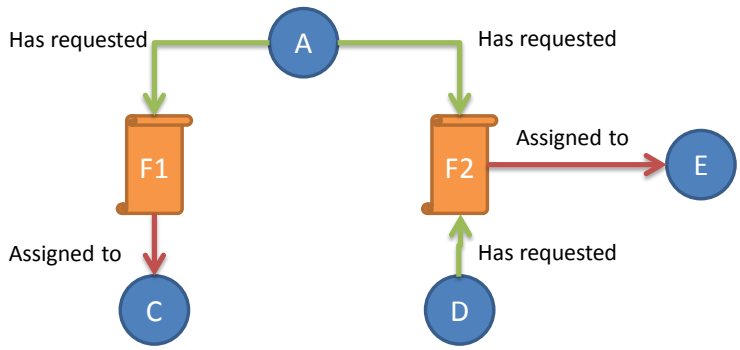


Starvation



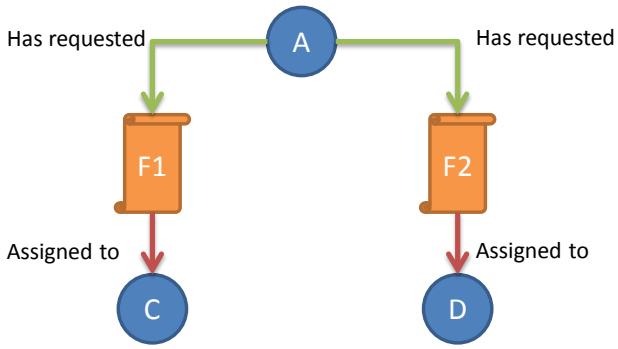
Process A needs files F1 and F2 to continue

Starvation



Process A still needs files F1 and F2 to continue

Starvation



Process A still needs files F1 and F2 to continue (starving)

Multitasking (Time-sharing)

- Extension of Multiprogramming
  - **Why?** **Need for user interactivity**
  - Instead of only switching jobs when waiting for IO, ***a timer can also cause jobs to switch***
- User interacts with computer via display and keyboard
  - Systems have to **balance CPU utilization against response time**
  - Better device management
- Need for ***file system*** to allow user to access data and code
- Need to provide user with an "**interaction environment**"

File System

- Method for ***storing and organizing computer files*** and the data they contain to make it easy to find and access them.
- File systems may use a data storage device such as a hard disk or CD-ROM and involve ***maintaining the physical location of the files***,
  - they might provide access to data on a file server by acting as clients for a **network protocol** (e.g., NFS, SMB, or 9P clients), or they may be **virtual** and exist only as an access **method for virtual data** (e.g., procfs, sysfs).
- More formally, ***a file system is a special-purpose database*** for the storage, organization, manipulation, and retrieval of data.

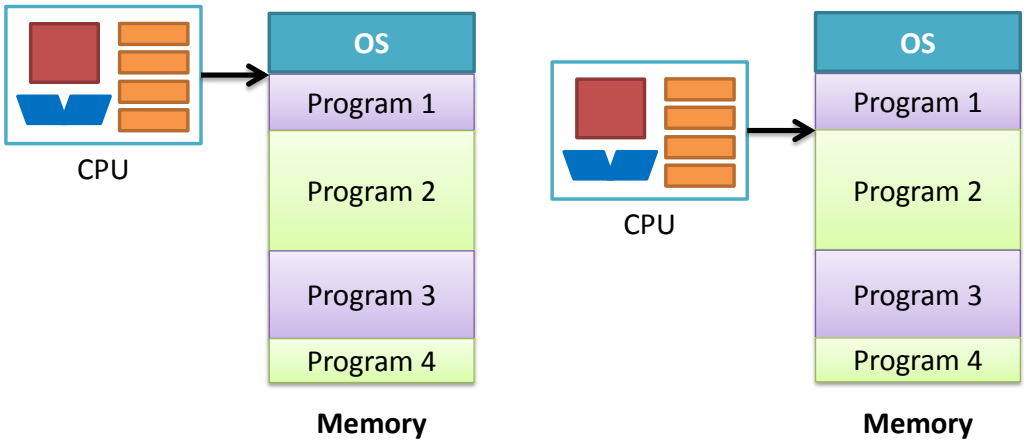
Memory Management

Operating systems must employ techniques to

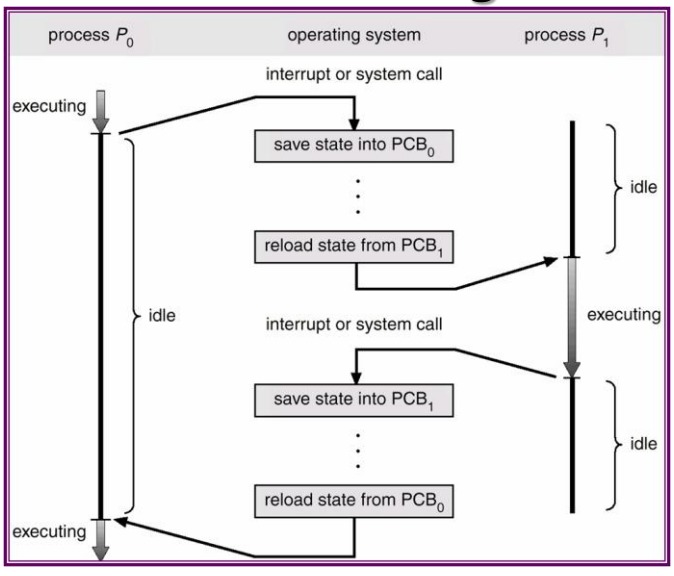
- Track **where and how a program resides in memory**
- Convert **logical addresses** into actual **addresses**

- **Logical address** (sometimes called a virtual or relative address) A value that specifies a generic location, ***relative to the program*** but not to the reality of main memory
- **Physical address** An actual address in the main ***memory device***

Partitioning



Time-Sharing

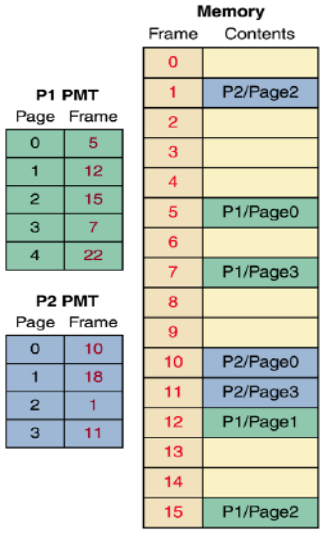


http://www.csie.ntnu.edu.tw/~swanky/os/chap4/CPU_Switch_From_Process_to_Process.png

Paged Memory Management

- **Paged memory technique** A memory management technique in which **processes** are divided into **fixed-size pages** and stored in **memory frames** when loaded into memory
  - **Frame:** A fixed-size portion of main memory that holds a process page
  - **Page:** A fixed-size portion of a process that is stored into a memory frame
- **Page-map table** (PMT) A table used by the operating system to keep track of page/frame relationships

Paged Memory Management



- To produce a physical address, you first look up the page **in the PMT to find the frame number** in which it is stored
- Then **multiply the frame number by the frame size** and **add the offset** to get the physical address

http://faculty.salina.k-state.edu/tim/ossg/Memory/paged_mem.html

Paged Memory Management

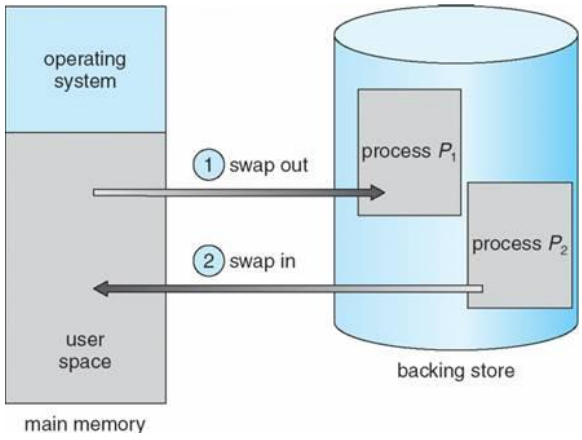
- **Demand paging** An important extension of paged memory management
  - Not all parts of a program actually have to be in memory at the same time
  - In demand paging, the pages are brought into memory on demand
- **Page swap** The act of bringing in a page from secondary memory, which often causes another page to be written back to secondary memory

Virtual Memory

- The demand paging approach gives rise to the idea of **virtual memory**, **the illusion that there are no restrictions on the size of a program**
  - *Virtual memory resides in disk, hence requiring a large access time.*
- Too much page swapping, however, is called **thrashing** and can seriously degrade system performance.
  - *It makes no sense to indefinitely increase the size of virtual memory*

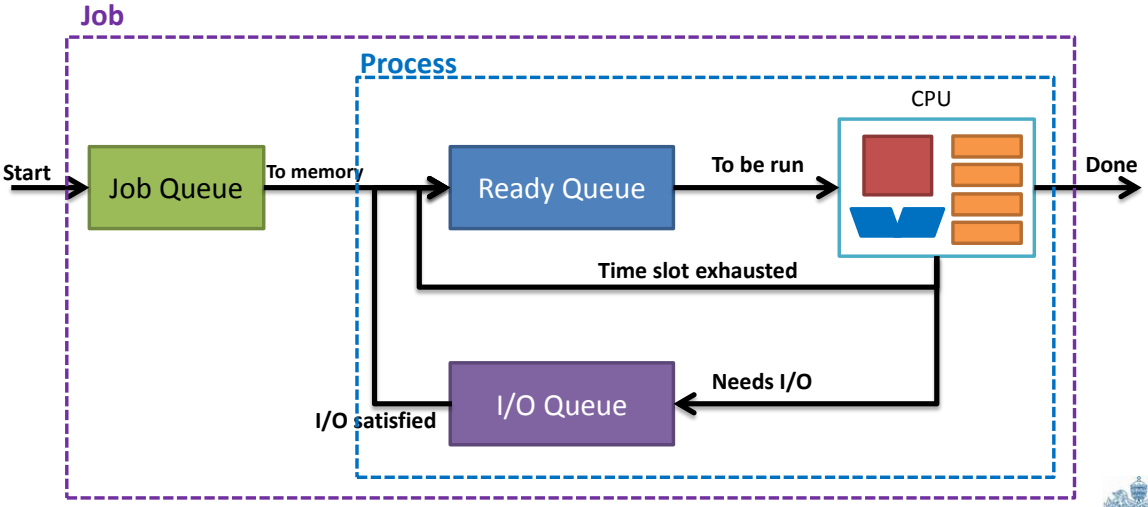
Virtual Memory

- Programs can be larger than memory
  - Program is ***loaded*** into memory as ***needed***
  - Active program and data are ***swapped*** to a disk until needed
- ***Memory space treated uniformly***



http://www.cs.odu.edu/~cs471w/spring12/lectures/MainMemory.htm

Queues for Process Management



CPU Scheduling

- **The act of determining which process in the ready state should be moved to the running state**
  - Many processes may be in the ready state
  - Only one process can be in the running state, making progress at any one time
  - ***Which one gets to move from ready to running?***
    - ***Any ideas???***

CPU Scheduling

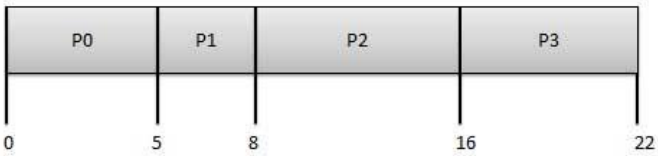
- Some concepts:
  - **Non-preemptive scheduling:** The currently executing process gives up the CPU voluntarily
  - **Preemptive scheduling:** The operating system decides to favor another process, preempting the currently executing process
  - **Turnaround time:** The amount of time between when a process arrives in the ready state the first time and when it exits the running state for the last time

CPU Scheduling Algorithms

- Some of the most famous algorithms:
  - **First-Come, First-Served (FCFS):** Processes are moved to the CPU in the order in which they arrive in the running state
  - **Shortest Job Next:** Process with shortest estimated running time in the ready state is moved into the running state first
  - **Round Robin:** Each process runs for a specified time slice and moves from the running state to the ready state to await its next turn if not finished

First-Come, First-Served

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16

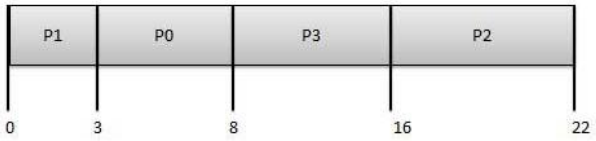


http://zannoos.blogspot.com.co/2015_01_01_archive.html

Shortest Job Next

Looks at all processes in the ready state and dispatches the one with the smallest service time

Process	Arrival Time	Execute Time
P0	0	5
P1	1	3
P2	2	8
P3	3	6

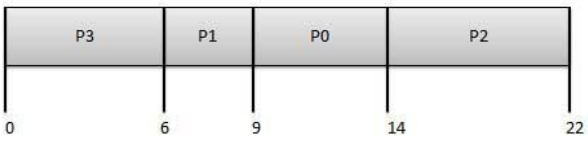


http://rannoos.blogspot.com.co/2015_01_01_archive.html

Priority Based Scheduling

- Each process is assigned a priority.
- Processes with same priority are executed on FCFS basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	0
P1	1	3	2	3
P2	2	8	1	8
P3	3	6	3	16



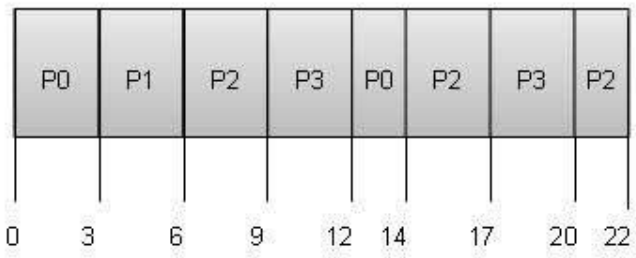
http://rannoos.blogspot.com.co/2015_01_01_archive.html

Round Robin

- ***Distributes the processing time equitably*** among all ready processes
- The algorithm establishes a particular time slice (or ***time quantum***), which is the amount of time each process receives before being preempted and returned to the ready state to allow another process its turn

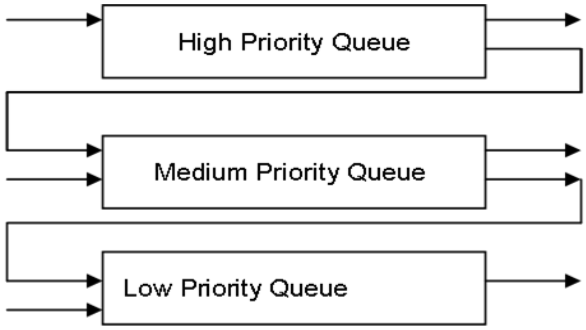
Round Robin

Quantum = 3



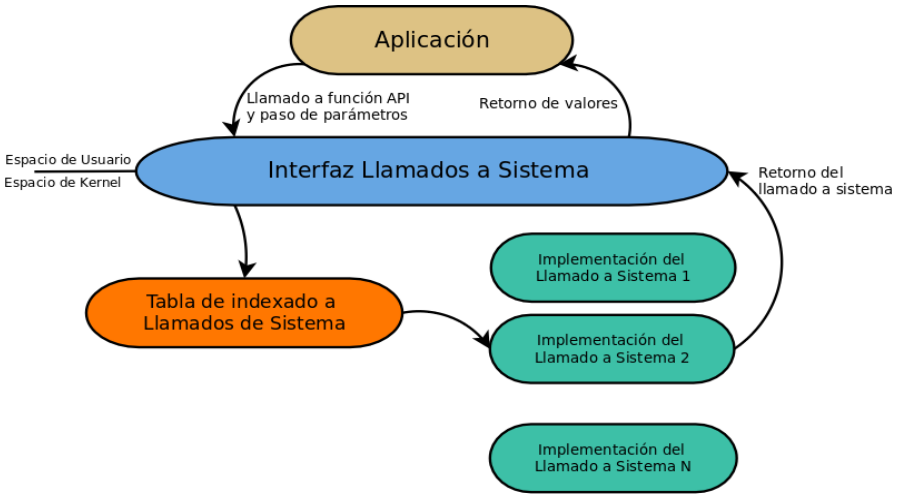
http://rannoos.blogspot.com.co/2015_01_01_archive.html

Adaptive Scheduling



http://www.read.cs.ucla.edu/111/_media/notes/mlfeedback.png

API – System Call – OS Relationship

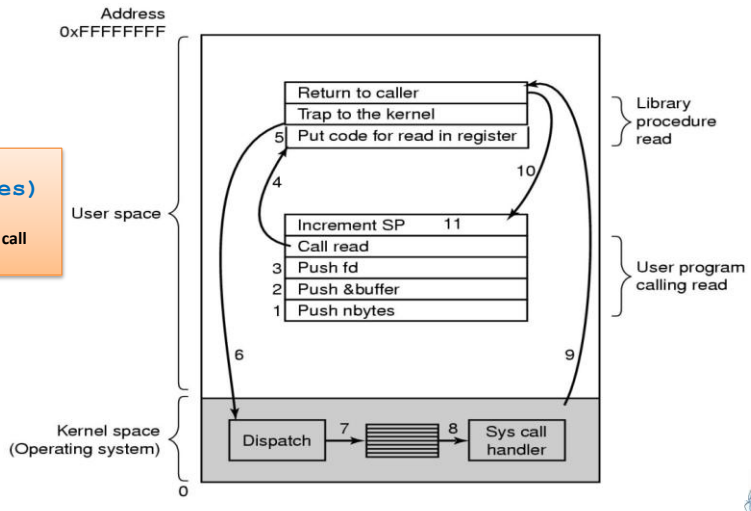


API (Application Programming Interface)

System Call

```
read (fd, buffer, nbytes)
```

There are 11 steps in making this system call



Some System Calls For Process Management and File Management

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

The meaning of some important folders

LINUX FILESYSTEM

Introduction

- **Filesystem**
  - How are data stored in storage?
  - How do users access the data? Data organization, files and directories
- **Filesystem types**
  - Disk FS: ext2, ext3, FAT, FAT32 & NTFS
  - Network FS: Samba & NFS
  - Flash FS: JFFS2
  - Special FS: proc FS

Introduction

- You should understand Linux FS
  - Why?
- **Everything in Linux is file, if it is NOT a process**
  - Easy to use
    - **Open file, read/write and close the file**
- Unlike Windows, Linux FS is a standard FS
  - Everyone should learn standards
- **Filesystem Hierarchy Standard (FHS)** started by **Dennis Ritchie**, 1993
- Defines the main directories and their contents in most Linux-based systems

FHS

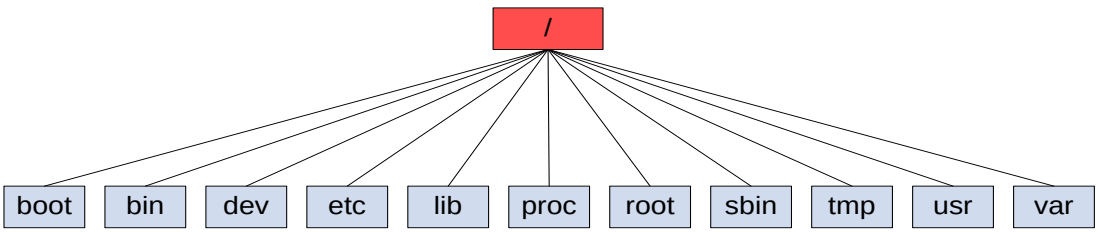
- There is not any drive C:, D:, …
- **All directories are under "/"**

  "/" is the root directory
- It is possible
  - to have **multiple partitions**
  - to have **multiple filesystems**

The "/"

- The primary hierarchy in FHS (filesystem hierarchy standard)
  - The root of tree of filesystem
- All paths start from here
- **There is only one "/" in filesystem**

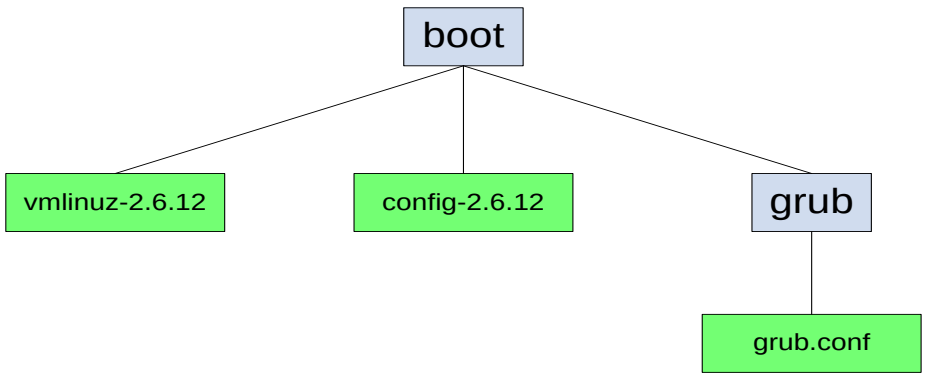
The "/"



boot

- **Linux kernel**
- **Boot loader configuration**
- If you lose **boot**
  - You cannot boot your OS

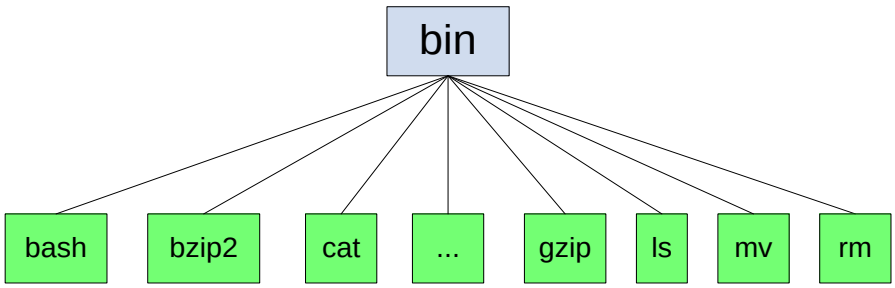
boot



bin

- **Essential programs**
- Need for system startup
- Basic commands for
  - Navigating in filesystem
  - File management

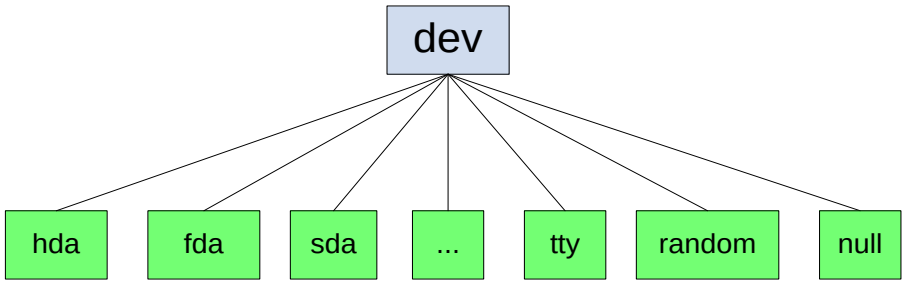
bin



dev

- Everything is file
  - **Hardware components (devices) are file**
    - Hard disk
    - Keyboard
- All device files are here
- **Direct interaction with device driver**
  - Open the device file
  - Read & Write

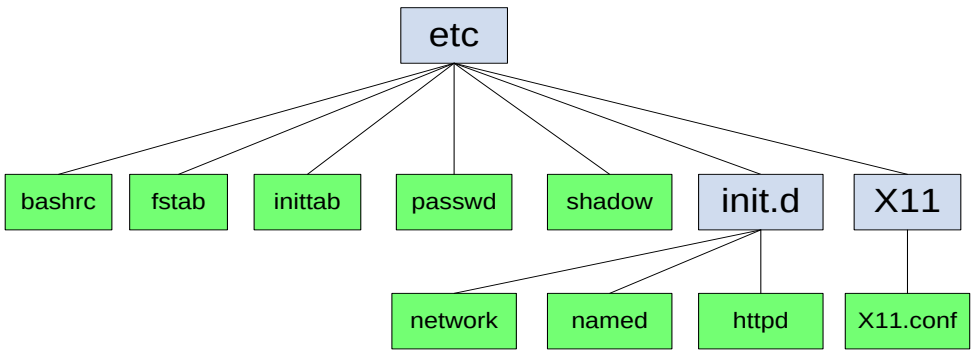
dev



etc

- **System configuration directory**
  - What is done by the registry in Windows
- **All configuration file are text files**
  - You can view and edit it manually

etc



home

- Home directory of user
- **Each user has a directory**

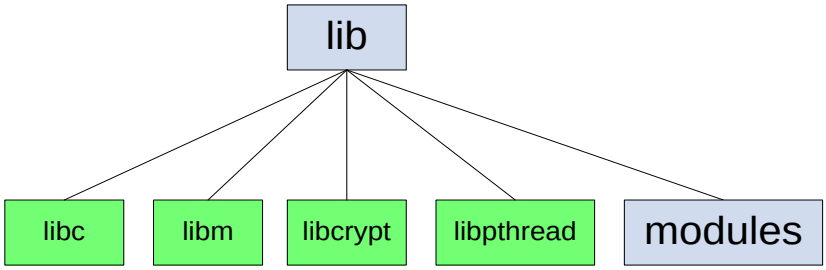
  `/home/hagar`

  `/home/diego`
- All user files are stored here

lib

- **Programs need libraries**
  - Dynamically linked libraries
- Programmers need libraries
- **All essential libraries are here**
  - Needed for system startup

lib



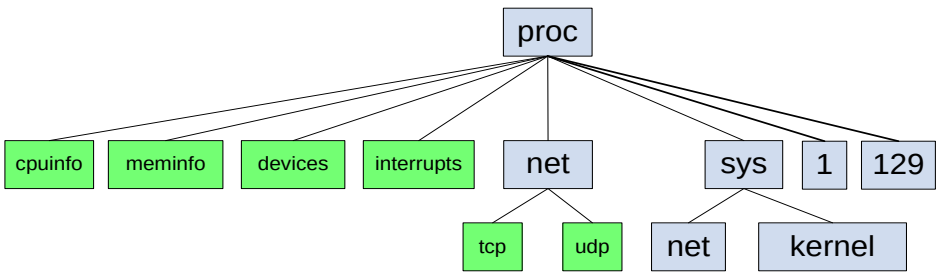
proc

- procfs (or the proc filesystem) is a special filesystem
  - **information about processes** and other system information in a hierarchical file-like structure
  - provides a more convenient and standardized method for dynamically **accessing process data held in the kernel**
- Typically, it is mapped to a mount point named **/proc** at boot time.

proc

- There is a directory for each running process, containing files that have useful information about the processes, such as
  - **exe**, which is a symbolic link to the file on disk the process was started from
  - **cwd**, which is a symbolic link to the working directory of the process
  - **wchan**, which, when read, returns the waiting channel the process is on
  - **maps**, which, when read, returns the memory maps of the process
- **/proc/cmdline** returns the command line passed to the running kernel
- **/proc/cpuinfo** returns information about the running CPUs
- **/proc/uptime** returns the uptime as two decimal values in seconds, separated by a space: the amount of time since the **kernel was started** and the amount of time that the **kernel has been idle.**
- **/proc/version** returns detailed information about the kernel; more than you'd get from **uname -a**

proc



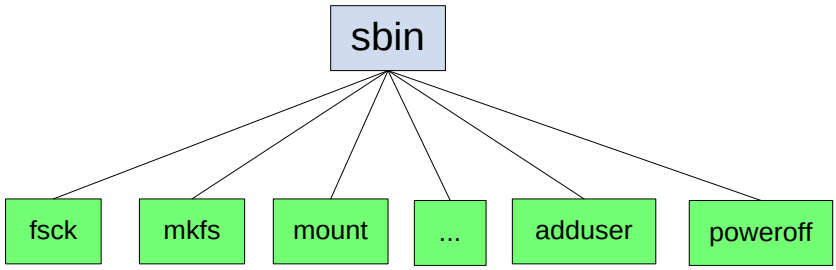
root

- **Home directory of root**
- Don't confuse
  - / is the "root of Filesystem"
  - root is the name of system admin
  - /root is the admin

sbin

- **System configuration programs**
  - Format hard disk
  - Manage hardware
- **Only "root" can run the programs**

sbin



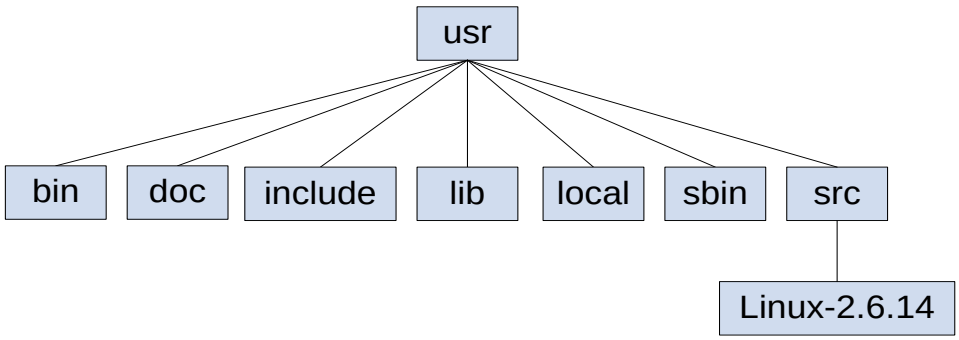
tmp

- Temporary directory
- All temp files created by programs
- Your temp files
- **It is emptied regularly!**

usr

- Very useful programs
  - **compiler, tools, etc.**
- Are not essential for system startup

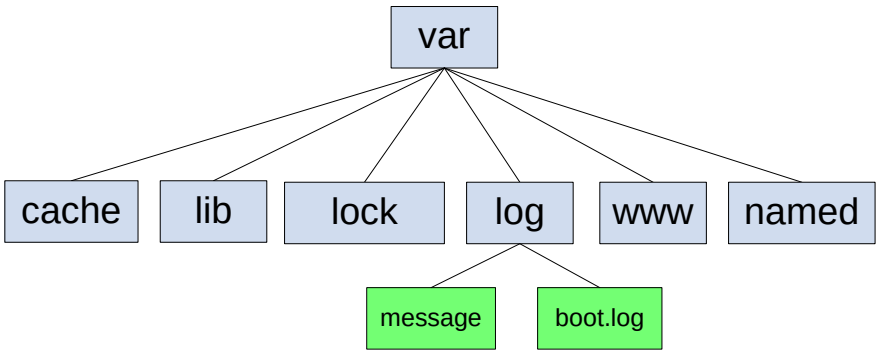
usr



var

- The variable directory
- All dynamic files
- User cannot change the files

var



Help

- Some documents are in **/usr/share/doc**
- Info pages are not complete help
  - **info <command name>**
- Man pages
  - /usr/share/man
  - **man <command name>**

Permissions Groups

Each file and directory has **three user-based permission groups:**

- **owner** - The Owner permissions apply only to the owner of the file or directory, they will not impact the actions of other users.
- **group** - The Group permissions apply only to the group that has been assigned to the file or directory, they will not affect the actions of other users.
- **all users** - The All Users permissions apply to all other users on the system, this is the permission group that you want to watch the most.

Permissions

- There are 3 basic permissions:
  - **Read (r), Write (w), Execute (x)**
- How to find them:
  - `ls -l`
- How to change them:
  - `chmod +/- r/w/x <filename>`

Permissions

- The permission in the command line is displayed as:

  `_rwxrwxrwx 1 owner:group`
- The first character is the special permission flag that can vary.

  _ (no special permissions), d (directory), l (symbolic link), among others
- The following set of three characters is for the **Owner.**
- The second set of three characters is for the **Group**.
- The third set of three characters is for the **All Users**.

Mounting

- Mount
  - **To add a filesystem to other filesystem**
  - Add you cool-disk FS to you laptop FS
- How?

```
mount <options> <device> <mount point>
mount -t vfat /dev/sdb1 /mnt/flash
```

- Don't forget the umount

```
umount <mount point>
umount /mnt/flash
```

Linux FS vs. Windows FS

- There is not drive C:, D:
- Top hierarchy is /
- **Path separator is / not **
- **File extensions have NOT any meaning**
- **There is not hidden attribute, hidden files are started by "."**

Basic Linux Commands

	Description	Example
cat	Sends file contents to standard output. This is a way to list the contents of short files to the screen. It works well with piping.	
	Sends the contents of the ".bashrc" file to the screen.	`cat .bashrc`
cd	Change directory	
	Change the current working directory to /home. The '/' indicates relative to root, and no matter what directory you are in when you execute this command, the directory will be changed to "/home".	`cd /home`
	Change the current working directory to httpd, relative to the current location which is "/home". The full path of the new working directory is "/home/httpd".	`cd httpd`
	Move to the parent directory of the current directory. This command will make the current working directory "/home.	`cd ..`
	Move to the user's home directory which is "/home/username". The '~' indicates the users home directory.	`cd ~`
cp	Copy files	
	Copy the files "myfile" to the file "yourfile" in the current working directory. This command will create the file "yourfile" if it doesn't exist. It will normally overwrite it without warning if it exists.	`cp myfile yourfile`
	With the "-i" option, if the file "yourfile" exists, you will be prompted before it is overwritten.	`cp -i myfile yourfile`
	Copy the file "/data/myfile" to the current working directory and name it "myfile". Prompt before overwriting the file.	`cp -i /data/myfile .`
	Copy all files from the directory "srcdir" to the directory "destdir" preserving links (-p option), file attributes (-p option), and copy recursively (-r option). With these options, a directory and all it contents can be copied to another directory.	`cp -dpr srcdir destdir`

http://www.comptechdoc.org/os/linux/usersguide/linux_ugbasics.html

Basic Linux Commands

	Description	Example
dd	Disk duplicate. The man page says this command is to "Convert and copy a file", but although used by more advanced users, it can be a very handy command. The "if" means input file, "of" means output file.	`dd if=/dev/hdb1 of=/backup/`
df	Show the amount of disk space used on each mounted filesystem.	
less	Similar to the more command, but the user can page up and down through the file. The example displays the contents of textfile.	`less textfile`
ln	Creates a symbolic link to a file.	
	Creates a symbolic link named symlink that points to the file test Typing "ls -i test symlink" will show the two files are different with different inodes. Typing "ls -l test symlink" will show that symlink points to the file test.	`ln -s test symlink`
locate	A fast database driven file locator.	
	This command builds the slocate database. It will take several minutes to complete this command. This command must be used before searching for files, however cron runs this command periodically on most systems.	`slocate -u`
	Lists all files whose names contain the string "whereis".	`locate whereis`
logout	Logs the current user off the system.	
ls	List files	
	List files in the current working directory except those starting with . and only show the file name.	`ls`
	List all files in the current working directory in long listing format showing permissions, ownership, size, and time and date stamp	`ls -al`

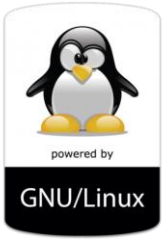
http://www.comptechdoc.org/os/linux/usersguide/linux_ugbasics.html

Basic Linux Commands

	Description	Example
more	Allows file contents or piped output to be sent to the screen one page at a time.	
	Lists the contents of the "/etc/profile" file to the screen one page at a time.	`more /etc/profile`
	Performs a directory listing of all files and pipes the output of the listing through more. If the directory listing is longer than a page, it will be listed one page at a time.	`ls -al |more`
mv	Move or rename files	
	Move the file from "myfile" to "yourfile". This effectively changes the name of "myfile" to "yourfile".	`mv -i myfile yourfile`
	Move the file from "myfile" from the directory "/data" to the current working directory.	`mv -i /data/myfile .`
pwd	Show the name of the current working directory	
	Lists the contents of the "/etc/profile" file to the screen one page at a time.	`more /etc/profile`
shutdown	Shuts the system down.	
	Shuts the system down to halt immediately.	`shutdown -h now`
	Shuts the system down immediately and the system reboots.	`shutdown -r now`
whereis	Show where the binary, source and manual page files are for a command	
	Locates binaries and manual pages for the ls command.	`whereis ls`

http://www.comptechdoc.org/os/linux/usersguide/linux_ugbasics.html

yocto PROJECT



Linux + Galileo

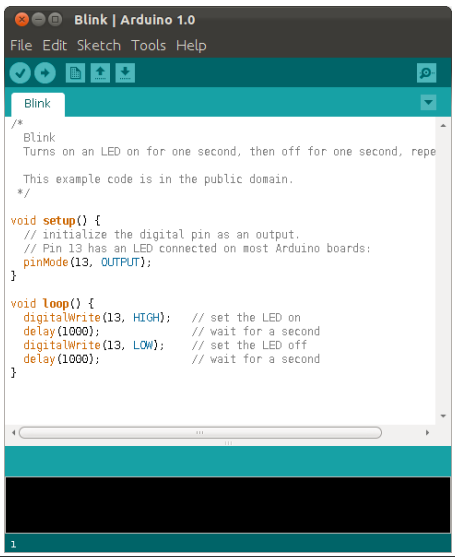
THE INTEGRATED DEVELOPMENT ENVIRONMENT

IDE

Options

1. **Arduino IDE** for Intel IoT Platforms
2. Intel **XDK** for IoT
3. Intel IoT **SDK** with eclipse
4. **Linux** development **on the boarD**

Arduino IDE for Intel IoT platforms



Problem #1 : Potential

- The original Arduino is based on a microcontroller. There's an **software emulator** simulating this microcontroller on Galileo and Edison.
- But Intel IoT platforms are **processor** based.
- That's why, with this method, you're only using a **minuscule fraction of the processing and networking potential of the platform.**

You can make IoT with this method,
but **basic IoT, not smart IoT**

Problem #2 : Compatibility

- This **software emulator** can't fully emulate the microcontroller, specifically the real time aspects.
- First because the OS is **not real time** itself, but also because the **granularity** would not be the same.

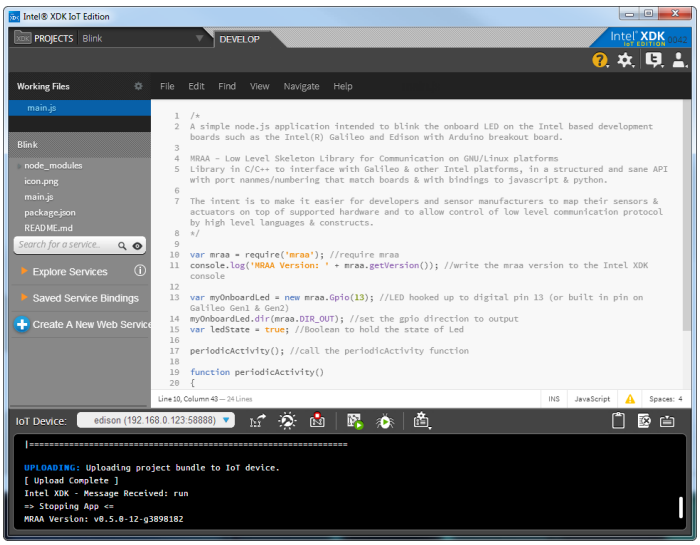
Result: **some core Arduino IDE libraries are not available**, some sensors won't work.

Intel XDK IoT Edition

- **Familiar for web developers** familiar with JavaScript.
  But if you like NodeJS on Linux, check the option 4 : linux.
- Limited functionalities and libraries but validated kits of **Plug'n'Play sensors**.
- Easy to install and start the IDE, but flashing the board is not yet easy. You'll need to find and **flash a microSD card.**

https://software.intel.com/en-us/iot/downloads

Intel XDK IoT Edition



Intel XDK IoT Edition

Public:

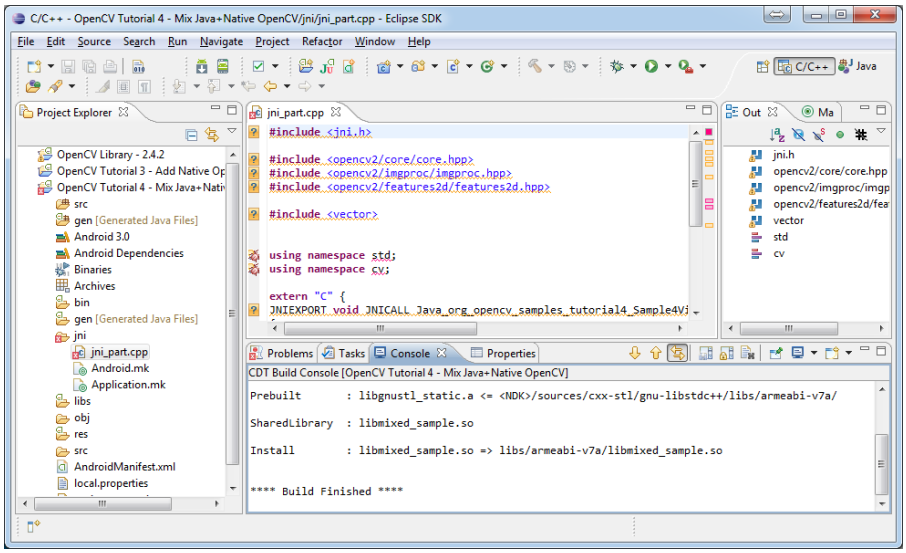
- Good for web devs or beginners in software development
- If your project plans to stay within the set of validated PnP sensors
- Not interested in using Linux at all

Compared to Arduino, you can add a lot more code with the Intel XDK. **You can evolve** into linux-NodeJS development if you want to go further.

Intel IoT SDK - Eclipse

- Eclipse on your workstation, communicating with the IoT board.
- **Mainly C** development.
- More control over the programming process and the **interaction with the OS**

Intel IoT SDK



Intel IoT SDK

Public :

- Fan of C development with Eclipse
- Don't need linux access
  or to add linux packages for your project
- Not interested in using linux at all
- Want to code high performance C code

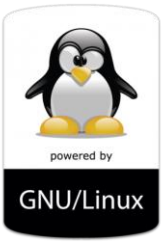
Compared to the XDK, **C is less fun than JavaScript, but more powerful**

Linux development on the board

- You **ssh** to the board from network or usb, **install all the packages** you need, and code with **any language on Linux**.
  - Nothing to install on your PC: Great for workshop setups.
  - Edit with **nano, emacs or vi** (you have to get used to that)
  - Compile with **gcc on the board**.
    - Or use VMs like Python, NodeJS, …
  - **Play with Linux services** like bluez for advanced Bluetooth features.
  - **Interact with the Yocto** professional embedded distribution, or even **rebuilt your distro** from scratch.

Not really the best way to develop

yocto PROJECT



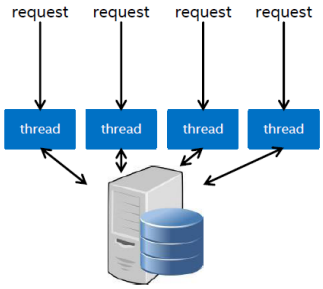
Linux + Galileo

THE INTEL XDK IoT EDITION

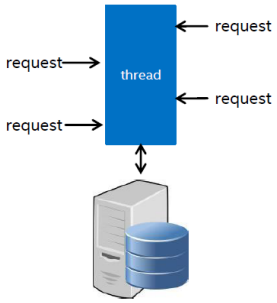
Node.js

What is Node.js?

- Node.js (or just *node*) **is a JavaScript runtime** designed for lightweight server applications
- It is **not a full webserver** (e.g. Apache, nginx)
- The **incoming request handler is single-threaded** instead of multi-threaded



Traditional server



node.js server

Why use Node.js?

- **Consistent:** Server/client language and data representations are the same
- **Scalable:** Single-threaded architecture minimizes memory usage and avoids cost of context-switching between threads
  - Problem: What if one client is computationally demanding?
- **Fast (at certain things)**
  - node is especially useful if I/O is likely to be your bottleneck (i.e., your server isn't doing that much)
  - examples: queued inputs, data streaming, web sockets
- Node.js is only an environment, **you have to do everything yourself.**
- Generally speaking, node is **ideal for *lightweight*, *real-time* tasks** and a **bad choice for *computationally intensive* tasks**

Working with Sensor Data

- Linux provides a virtual filesystem called **sysfs** that allows for easy access to underlying hardware from userspace
- This makes working with sensor data **as simple as reading and writing to files**
  - ***Arduino functionality on Galileo is implemented via abstracted sysfs interactions***
- Quick example: reading the core temperature

  ```
  cat /sys/class/thermal/thermal_zone0/temp
  ```

  Divide by 1000 to get the SoC temperature in ^{o}C
- (Quark can run hot, but it's normal)

Working with Sensor Data – GPIO access

- Export the port

  ```
  echo -n "3" > /sys/class/gpio/export
  ```

  – A new folder (gpio3) will appear in /sys/class/gpio
  – This particular GPIO pin is wired to the green onboard LED
- Set port direction

  ```
  echo -n "out" > /sys/class/gpio/gpio3/direction
  ```
- Read/write value

  ```
  echo -n "1" > /sys/class/gpio/gpio3/value
  echo -n "0" > /sys/class/gpio/gpio3/value
  ```

Back on Node.js

- There is a node module called *fs*to handle filesysteminteractions

```
fs.readFile('/etc/passwd', function (err, data) {
    if (err) throw err;
    console.log(data);
});
```

- We could use this to handle all GPIO interactions, but **there is a nice npm** (node package manager) **wrapper called *galileo-io*** to make this a little cleaner
  - This is only capable of digital read/write and analog read/write from individual pins
  - Other useful Galileo hardware requires a bit more (UART, I2C, SPI, etc)

MRAA and UPM

- The **Yocto system** installed on the Galileo board **provides a set of libraries** especially designed for the Intel XDK IoT Edition.
- These libraries are **MRAA** and **UPM**.
- MRAA (or libmraa) is a low-level library that offers a translation from the General Purpose Input/Output (GPIO) interfaces to the pins available on the Galileo board.
- So instead of reading the raw level information from the GPIO module available on the Linux kernel, a developer can easily select a pin number and work directly with it.
- MRAA takes care of the underlying details.

MRAA - Low Level Skeleton Library for Communication on GNU/Linux platforms
https://github.com/intel-iot-devkit/mraa - http://iotdk.intel.com/docs/master/mraa/

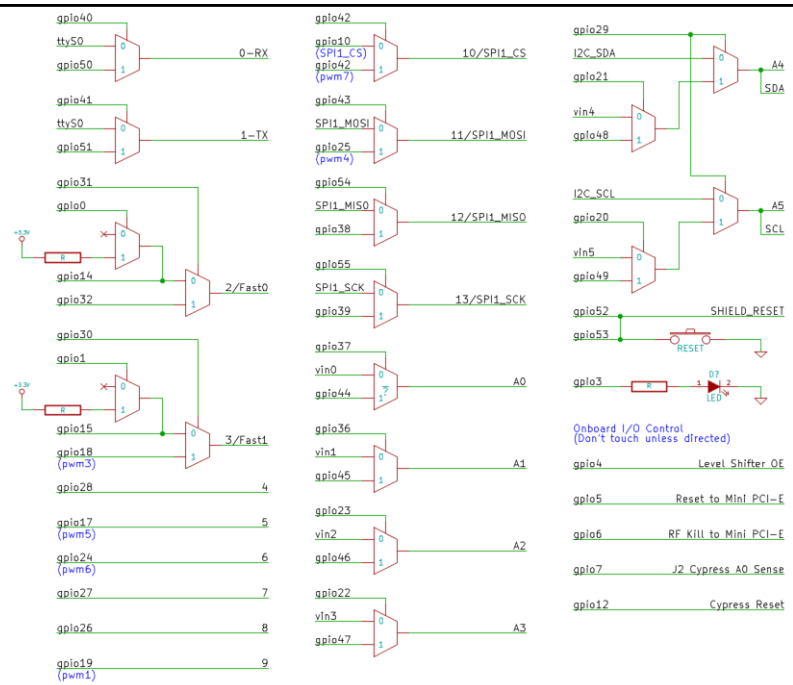
UPM (Useful Packages & Modules) Sensor/Actuator repository for MRAA
https://github.com/intel-iot-devkit/upm - http://iotdk.intel.com/docs/master/upm/

MRAA and UPM

- **UPM (or libupm) is a repository of sensor representations**, written in C++ and utilizing MRAA.
- Both the UPM and MRAA libraries have **C++ bindings to JavaScript**.
- With the Intel XDK IoT Edition, **you use Node.js to communicate with all of the GPIO pins, libraries, and packages.**
- For instance, you can easily create a code representation of GPIO pin 3 (onBoard LED) with a useful line of code like

```
var myOnboardLed = new mraa.Gpio(3, false, true);
```

GPIO Pins Assignment



http://www.malinov.com/Home/sergey-s-blog/intelgalileo-programminggpiofromlinux

Getting Started Projects

1. **Setting up the Board**
   - Creating a bootable SD card with the Linux image
   - Installing and starting the Intel XDK and connecting with the board
   - Basic Linux commands
2. **OnBoard LED blink (fixed frequency)**
3. **Digital Read and Digital Write**
   - Button changes state of LED
4. **Analog Read**
   - Read an analog input (temperature) and print it on the console
5. **Pulse-Width Modulation (PWM)**
   - Control an LED intensity (PWM) depending on the light intensity in the room
6. **LCD display**
   - Show the temperature on the LCD display and modify the background color if the temperature is above certain threshold
7. **Simple WebServer**
   - Display the temperature and toggle an LED through a webpage
8. **Advanced WebServer**
   - Display the temperature using a tachometer-like graph and control a servo-motor

References

- **INTRODUCTION TO THE INTERNET OF THINGS.** Mirko Franceschinis (franceschinis@ismb.it). Workshop on Scientific Applications for the Internet of Things (IoT). 16-27 March 2015, ICTP – Trieste. Pervasive Technologies (PerT) research area. Istituto Superiore Mario Boella.
- **GALILEO DATASHEET OVERVIEW.** Website: www.intel.com
- **EMBEDDED LINUX BOARD COMPARISON.** Raspberry Pi, Beaglebone Black, Arduino Yun, and Intel Galileo--which one is right for you? Adafruit website: https://learn.adafruit.com/embedded-linux-board-comparison/overview
- **SEEEDSTUDIO GROVE STARTER KIT FOR INTEL GALILEO GEN 1**. SeeedStudio website: http://www.seeedstudio.com/depot/Grove-starter-kit-plus-Intel-IoT-Edition-for-Intel-Galileo-Gen-1-p-1977.html
- **10 GREAT INTEL GALILEO FEATURES.** Makezine website: http://makezine.com/2013/10/03/10-great-intel-galileo-features/
- **THE INTERNET OF THINGS.** MIT Technology Review. Business Report. July/August 2014
- **INTEL IoT INSIGHTS 2014.** Intel website: www.intel.com/IoT
- **SENSORS: ANALOG & DIGITAL.** Jonathan Brewer (jon@nsrc.org). Network Startup Resource Center (NSRC). University of Oregon. Workshop on Scientific Applications for the Internet of Things (IoT). 16-27 March 2015, ICTP – Trieste.
- **ITP PHYSICAL COMPUTING.** Online course. NYU website: https://itp.nyu.edu/physcomp/itp/syllabus/
- **THE INTERNET OF THINGS WITH INTEL.** Intel Academic IoT Course. Paul Guermonprez. Toulouse, France. Github website: https://github.com/guermonprez/intel-academic-IoT-course
- **OPERATING SYSTEM STUDY GUIDE.** Course website: http://www.csie.ntnu.edu.tw/~swanky/os/
- **OPERATING SYSTEM – MULTI-THREADING.** Website: http://rannoos.blogspot.com.co/2015/01/operating-system-multi-threading.html
- **OPERATING SYSTEMS STUDY GUIDE. K-State Salina. Website: http://faculty.salina.k-state.edu/tim/ossg/index.html**
- **COURSE: CSE4/521: OPERATING SYSTEMS**. Bina Ramamurthy, State University of New York at Buffalo. Website: http://www.cse.buffalo.edu/~bina/
- **OPERATING SYSTEMS.** Brooks Cole, Slides. 2003
- **COMPUTER SCIENCE AN OVERVIEW**. Gleen Brookshear. Pearson Addison-Wesley. Eight Edition. 2005
- **CHAPTER 10: OEPRATING SYSTEM.** Dr. J. Carette, CS 1MD3 T2 2005. Website: http://www.cas.mcmaster.ca/~carette/CS1MD3/2005/slides.html)
- **COMPUTERS AS COMPONENTES, PRINCIPLES OF EMBEDDED COMPUTER SYSTEM DESIGN**. Wayne Wolf. Elsevier. 2005
- **GALILEO TUTORIAL NETWORKING AND NODE.JS**. Senzations 2014. Jason Wright. Website: https://github.com/jpwright/senzations14-galileo-nodejs
- **MRAA** – Low Level Skeleton Library for Communication on GNU/Linux platforms. Website: https://github.com/intel-iot-devkit/mraa http://iotdk.intel.com/docs/master/mraa/
- **UPM** (Useful Packages & Modules) Sensor/Actuator repository for MRAA. Website: https://github.com/intel-iot-devkit/upm http://iotdk.intel.com/docs/master/upm/