http://www.isen.fr/

Introduction to development of embedded Linux systems

Julien Gaulmin

<julien23@gmail.com> / @julien23

Version 2015r2.

This course is freely distributable under the terms of the *Creative Commons License*
(http://creativecommons.org/licenses/by-sa/2.0/fr/deed.en)
Attribution-ShareAlike 2.0

Summary

1. Embedded computing:
    - Definitions,
    - Market and prospect,
    - Embedded system topology,
    - Hardware architecture,
    - Software architecture.
2. Why GNU/Linux?
    - Technological reasons,
    - Economic reasons,
    - Personal reasons,
    - Other OS,
    - Licenses,
    - Limits.

3. Solutions:
   - Types of solutions,
   - Product oriented platforms,
   - Base software components,
   - References.
4. Essentials:
   - Unix concepts and orthodoxy,
   - Linux boot process analysis,
   - Compilation process,
   - Binary link edition,
   - Executables,
   - μClinux vs Linux.

5. Methods and development tools:
    - Terminology,
    - Development method,
    - Cross-compilation,
    - Optimisation and debug,
    - Software emulation and virtualization.
6. Case study;
7. References.

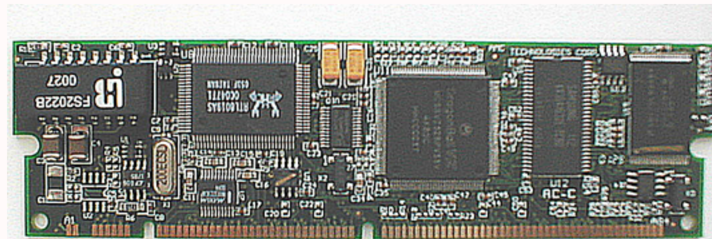
Embedded computing

Definitions

- Software and hardware combination dedicated to one or more duty according to more or less severes **constraints** (consumption, temperature, size, performances...);
- **Autonomous / Self-governing** software and hardware combination missing **traditional inputs/outputs** (screen, keyboard, mouse...);
- Hidden computer **integrated** in a system or equipment not relative to a computing task;
- More generaly can describe all the electronic systems which are not members of the traditional computing areas (desktop, web, business, big data);
- We also talk about **"buried"** or deeply embedded systems when the link with computing is not clear anymore.

Exemples

- Smartphone, tablet, diskless-PC, video game console;
- TV box, NVR, camera, car computer;
- Industrial machine, robot;
- Router, Inernet box, μCsimm, Raspberry Pi;
- Washing machine, ABS.



Real-time and embedded computing

- Real-time system:
  - data after capture and processing is still **relevant**,
  - ability to respond to a request/stimulus and to produce an appropriate response in a given time $\Rightarrow$ **determinism**,
  - not necessarily synonymous with computing power or speed of execution.
- Some embedded systems are subject to more or less strong temporal constraints requiring the use of real-time kernels (RTOS[1]);

[1] RTOS: ***R****eal* ***T****ime* ***O****perating* ***S****ystem*

- Two main forms of real-time applications:
  - **hard real-time** $\Rightarrow$ the system **must** respond to a given event in a **given time** (ABS, military system...),
  - **soft real-time** $\Rightarrow$ the system is subject to **temporal constraints** but the delay or cancellation of a deadline is not a big deal (video game, VoIP[1]...).

[1] VoIP: *Voice over **IP***

Market and prospect

The embedded rise

- Coupled with the rise of the "all-digital" and multimedia;
- Convergence of media (voice, video, data ...)
- Intelligence at all levels (automation, robotics...)
- Connected products, mobility;
- Miniaturization and lower component costs;
- The next growing factors : IoT[1], wearables, etc;
- Evolution of the total embedded market from $32M in 1998 to $92M in 2008 and $2000M in 2015[2].

[1] IoT: *Internet of Things*
[2] Estimation from IDC in 2011

Customs and habits

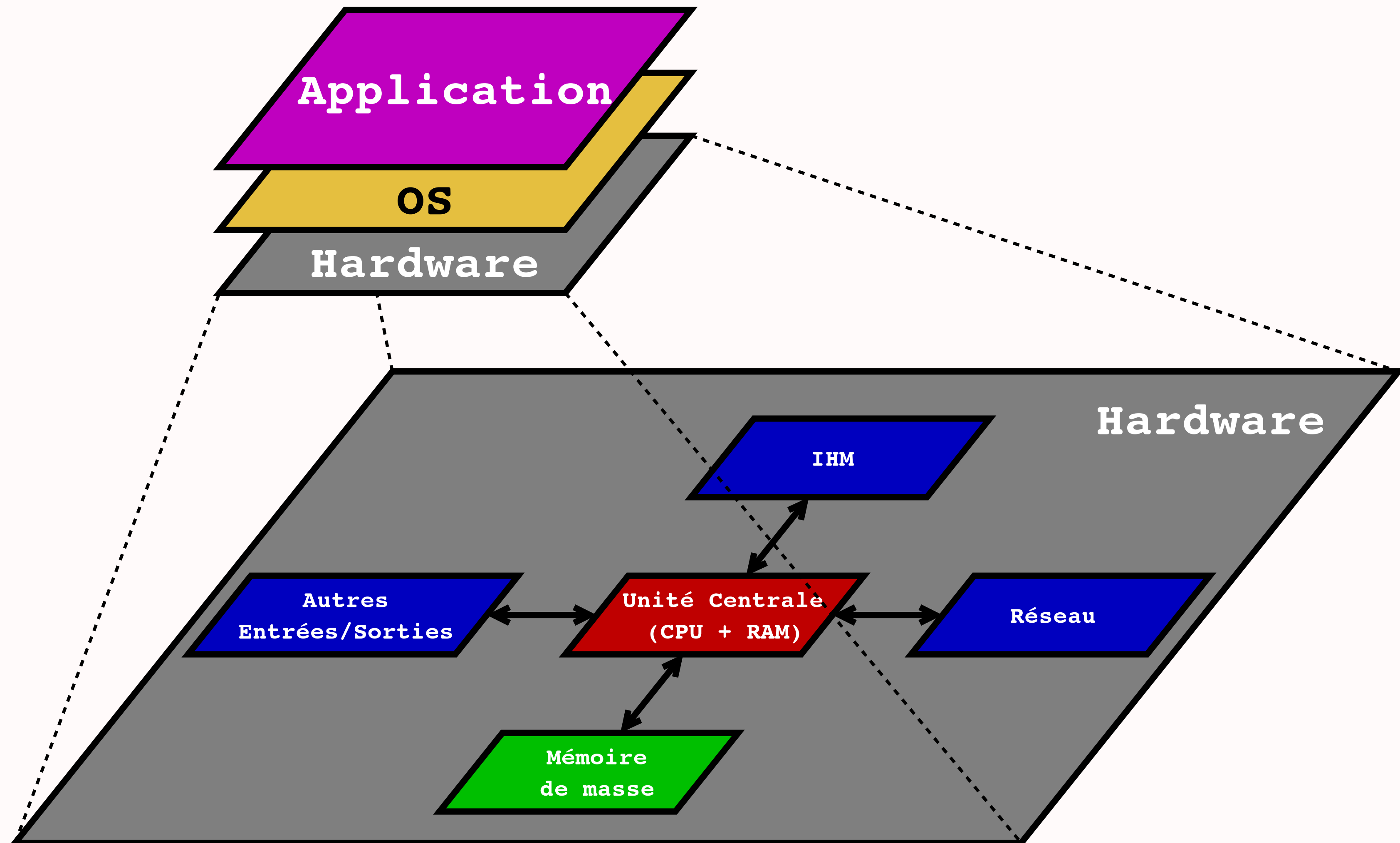
- ✗ Closed market of proprietary OS:
  - not cross-compatible,
  - expensive and frozen development kits,
  - high royalties,
  - dependence of an editor.
- ✗ "Home made" OS:
  - long development and expensive maintainability,
  - bad scalability and sustainability,
  - reduced portability.
- ✓ Since 2000, Linux and free software have emerged as an alternative to these monopolies, from prototyping to finished product.

The players

- Developers communities;
- Software publishers;
- Service companies;
- Manufacturers components;
- Industrials;
- Scientists, academics and students;
- Organizations (CELF[1], Linux Foundation, TV Linux Alliance, RTLF[2], LDPS[3], FHSG[4], LSB[5], FSF[6], OpenGroup...)
- Medias (web portals, editors, press...).

[1]CELF: ***CE L**inux **F**orum*
[2]RTLF: ***R**eal-**T**ime **L**inux **F**oundation*
[3]LDPS: ***L**inux **D**evelopment **P**latform **S**pecification*
[4]FHSG: ***F**ilesystem **H**ierarchy **S**tandard **G**roup*
[5]LSB: ***L**inux **S**tandard **B**ase*
[6]FSF: ***F**ree **S**oftware **F**oundation*

Embedded system topology



Hardware architecture

- Often dedicated in systems with severe consumption, size or cost constraints;
- Today, the trend was reversed with the emergence of increasingly integrated *system off-the-shelves* (SOB[1], SOC[2]...);
- Suited to the needs $\Rightarrow$ no superfluous (scale savings).

[1] SOB: ***S****ystem* ***O****n* ***B****oard*
[2] SOC: ***S****ystem* ***O****n* ***C****hip*

CPU families

- **General purpose**: x86, ia64, x64, PowerPC, Sparc...
- **Low power**: ARM[1] (ARMx, Cortex, XScale), SuperH, MIPS[2], PowerPC...
- **SOC**: 68k (Motorola DragonBall et ColdFire), x86 (AMD Geode, VIA Nano, Intel Atom), ARM (NVidia Tegra, Qualcomm Snapdragon, Samsung Hummingbird, Apple Ax, Intel PXA), MIPS, PowerPC, Etrax...
- **ASIC or FPGA** with ARM, MIPS or PowerPC core(s)...

---

[1] ARM: ***A****dvanced* ***R****ISC* ***M****achine*

[2] MIPS: ***M****icroprocessor without* ***I****nterlocked* ***P****ipeline* ***S****tages*

Which of the following 32-bit chip families would you consider for your next embedded project?

Chip family	2012 (N = 1,548)	2011 (N = 1,679)
ARM Cortex/ARM9/ARM11	38 %	47 %
STMicroelectronics STM32 (ARM)	24 %	20 %
TI Stellaris (ARM)	21 %	10 %
Microchip PIC 32-bit (MIPS)	21 %	18 %
TI OMAP (ARM)	20 %	24 %
Atmel AT91xx (ARM)	17 %	17 %
Atmel (AVR32)	16 %	15 %
Intel Atom, Pentium, Celeron, Core IX	15 %	19 %
NXP ARM	15 %	16 %
Freescale DragonBall i.MX (ARM)	14 %	7 %
Freescale Kinetis (Cortex-M4)	14 %	13 %
TI Sitara (ARM)	11%	
Xilinx MicroBlaze (soft-core)	11%	11%
Altera Nios II (soft core)	9 %	12 %
Arduino	9 %	
Altera SoC-FPGA (ARM)	9 %	
Freescale 68K, ColdFire	9 %	13 %
TI C2000 MCUs	8 %	
Freescale PowerPC 55xx	8 %	8 %
Freescale PowerPC 7xx, 8xx	7 %	9 %
Renesas SuperH, H8SX, M32C, M32R	7 %	6 %
Freescale PowerQUICC	7 %	8 %
Cypress PSOC 5 (ARM)	7 %	8 %
AMD Fusion, Athlon, Opteron, Geode	6 %	6 %

Chip family	2012 (N = 1,548)	2011 (N = 1,679)
Freescale PowerPC 5xx, 6xx	6%	8%
Xilinx Virtex-5 (with PowerPC 405)	6%	8%
TI Hercules (ARM)	6%	
Actel/Microsemi ProASIC 3 (ARM)	6%	4%
Xilinx Zynq	6%	
Marvell	5%	6%
NVIDIA Tegra	5%	
Xilinx Virtex-4 (with PowerPC 405)	4%	5%
Broadcom (any)	4%	
IBM PowerPC 4xx, 7xx	3%	4%
NEC V850	3%	2%
Qualcomm (any)	3%	
STMicroelectronics ST20	3%	2%
Intel Itanium	3%	3%
Energy Micro EFM32	3%	3%
Cirrus Logic EP73xx, EP93xx (ARM)	2%	2%
Infineon Tricore	2%	2%
AMD Alchemy (MIPS)	2%	2%
AMCC PowerPC 4xx	2%	3%
SPARC (any)	1%	1%
Fujitsu FR series	1%	1%
IDT 32xxx	%	%
Other	6%	5%

Copyright © 2012 by UBM Electronics. All rights reserved.

Communication bus

- VME[1] ⇒ VMEbus, VME64, VME64x, VME320, VXI[2], IP-Module, M-Module...
- PCI[3] ⇒ CompactPCI (cPCI), PCI-X, PXI[4], PMC[5], PC/104+, PCI-104, MiniPCI...
- PCIe[6] ⇒ XMC, AdvancedTCA, AMC, ExpressCard, MiniCard, PCI/104-Express, PCIe/104...
- PCMCIA[7] ⇒ PCMCIA, PC Card, CardBus...
- ISA[8] ⇒ PC/104...

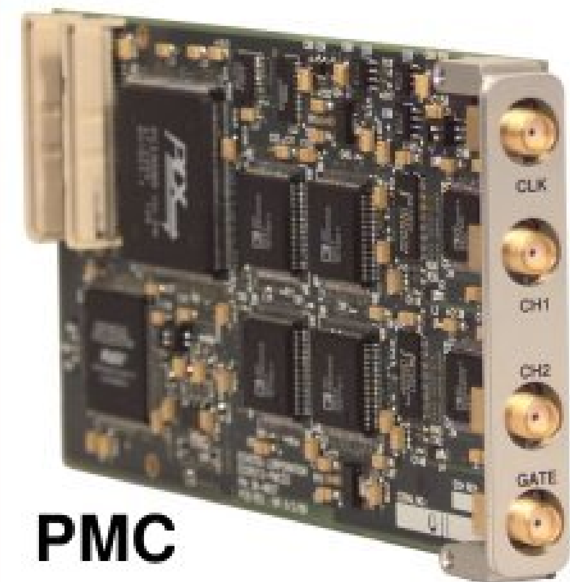
---

[1]VME: ***V**ersa **M**odule **E**urocard*
[2]VXI: ***VME**bus e**X**tension for **I**nstrumentation*
[3]PCI: ***P**eripheral **C**omponent **I**nterconnect*
[4]PXI: ***PCI** e**X**tension for **I**nstrumentation*
[5]PMC: ***PCI M**ezzanine **C**ard*
[6]PCIe: ***PCI e**XPRESS*
[7]PCMCIA: ***P**ersonnal **C**omputer **M**emory **C**ard **I**nternational **A**ssociation*
[8]ISA: ***I**ndustry **S**tandard **A**rchitecture*

VME

PC/104+

PXI
PDXI
PowerDAQ


CLK
CH1
CH2
GATE
PMC

- Parallel ⇒ ATA/ATAPI[1] (IDE[2]), SCSI[3], Centronics/IEEE1284...
- Série ⇒ I^2C[4], RS232, RS485, USB[5], IEEE1394, Serial ATA...
- Network ⇒ Ethernet, FDDI[6], X.25, WiFi/802.11, BlueTooth/ZigBee/WUSB/Wibree/802.15.x/ANT, IrDA[7], ATM[8], Token Ring, GSM[9]/GPRS[10]/UMTS[11]/LTE[12]...

---

[1] ATAPI: ***AT A**ttachment **P**acket **I**nterface*
[2] IDE: ***I**ntergated **D**rive **E**lectronics*
[3] SCSI: ***S**mall **C**omputer **S**ystems **I**nterface*
[4] I^2C: ***I**nter-**I**ntegrated **C**ircuit*
[5] USB: ***U**niversal **S**erial **B**us*
[6] FDDI: ***F**ibber **D**istributed **D**ata **I**nterface*
[7] IrDA: ***I**nf**r**ared **D**ata **A**ssociation*
[8] ATM: ***A**synchronous **T**ransfert **M**ode*
[9] GSM: ***G**lobal **S**ystem for **M**obile communications*
[10] GPRS: ***G**eneral **P**acket **R**adio **S**ervice*
[11] UMTS: ***U**niversal **M**obile **T**elecommunications **S**ystem*
[12] LTE: ***L**ong **T**erm **E**volution*

Mass storage

- Magnetic storage ⇒ 2,5", 3,5", microdrive, tape...
- Flash memory ⇒ SSD[1], FlashDisk, DiskOnChip, CompactFlash, CFI[2], SD Card, Memory Stick, USB Mass Storage...
- ROM[3], EPROM, EEPROM, UVPROM...
- Optical storage ⇒ CD, DVD, Blu-ray...
- Combination of the above.

[1] SSD: ***S**olid **S**tate **D**evice*
[2] CFI: ***C**ommon **F**lash **I**nterface*
[3] ROM: ***R**ead **O**nly **M**emory*

I/O

- Inputs:
  - AON[1] (open collectors, optocouplers...) or GPIO[2],
  - sensors/transducers (pressure, sound, temperature...),
  - keyboards, buttons, touch screens, remote controls (infrared, radio...)
  - optical sensors (photo/video), radio readers (tags), laser readers (barcodes).
- Outputs:
  - AON (relay, optocouplers, logic...) or GPIO,
  - LEDs, screens and displays,
  - beeps, speech, alarms,
  - all kind of printers (paper, labels, photos...).

[1] AON: **A**ll **O**r **N**othing

[2] GPIO: ***G**eneral **P**urpose **I**nput **O**utput*

Network

- Technologies:
  - classical ⇒ Ethernet, ATM, X.25...
  - fieldbus ⇒ CAN[1], RS232, RS485, PLC[2], ARCnet[3],
  - wireless ⇒ WiFi/802.11, IrDA, BlueTooth/ZigBee/WUSB/Wibree/802.15.x, GSM/GPRS/UMTS...
- Why?
  - communicate,
  - share informations or status,
  - monitor and control.

---

[1]CAN: ***C**ontroller **A**rea **N**etwork*

[2]PLC: **P**ower **L**ine **C**ommunication

[3]ARCnet: ***A**ttached **R**essource **C**omputer **net**work*

Software architecture

Reminder about operating systems (OS)

- Composed of a kernel and some drivers that make the **hardware abstraction**;
- Of **libraries** which formalize the API[1] to access the kernel services;
- And a variable set of **basic tools** (hardware setup, file management, GUI, etc).

[1] API: *Application Programming Interface*

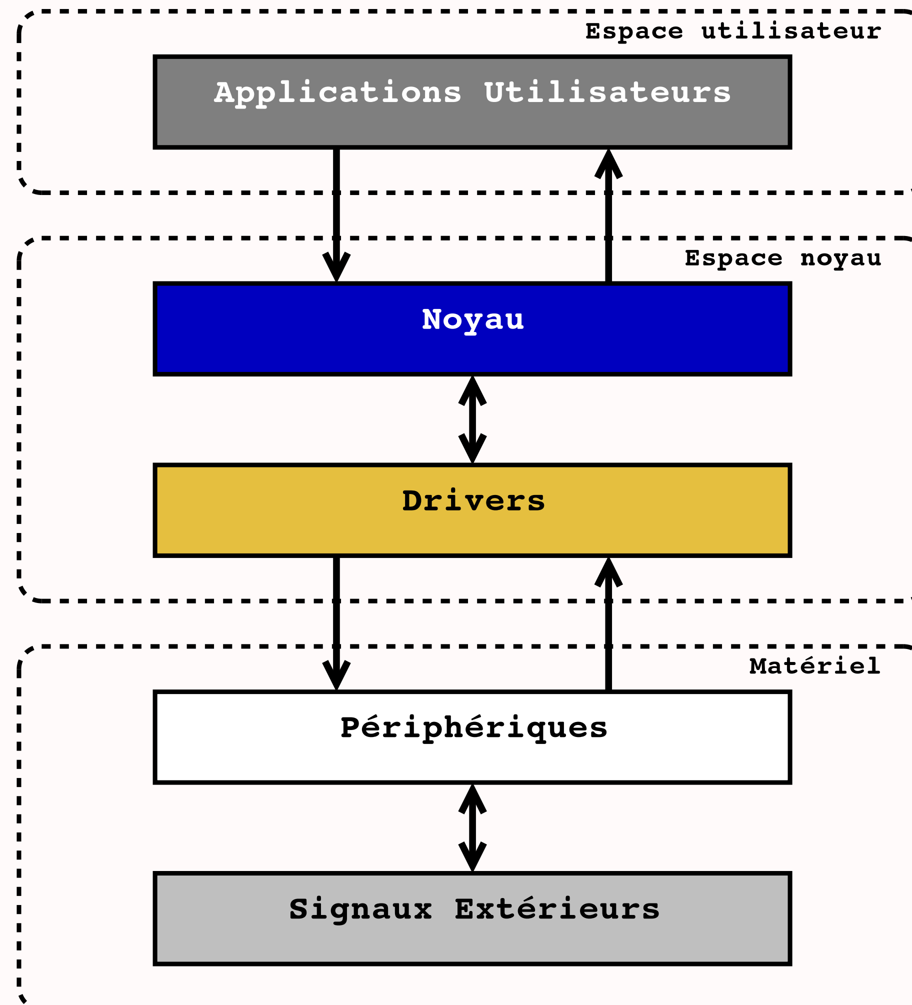
The kernel

- First program executed after startup;
- Typically uses a privileged mode of execution of the CPU;
- Performs a **hardware and services abstraction** :
  - provides a suite of general services that facilitate the creation of application software,
  - serves as an intermediary between software and hardware,
  - brings convenience, efficiency and scalability,
  - for introducing new features and new equipment without affecting existing software.

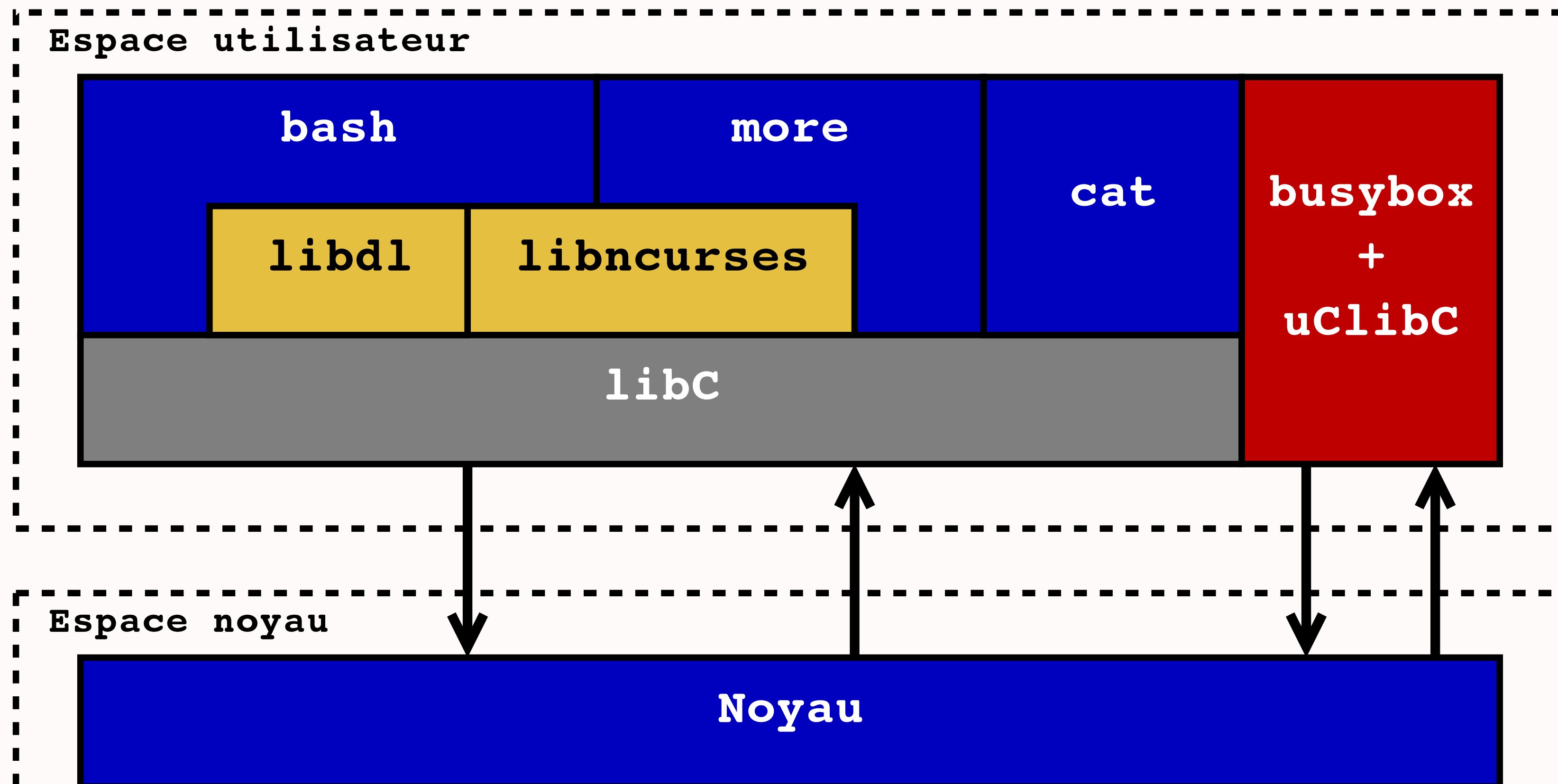
- The features offered differ from one model to another and are typically related to:
  - running and **scheduling softwares**,
  - managing **main memory and peripherals**,
  - handling and organization of files (**filesystems**),
  - communication and network, and
  - error detection.

Source: Wikipedia (http://http://en.wikipedia.org/wiki/Operating_system)

Structure of a monolithic OS



Structure of the user space



Why GNU/Linux?

Technological reasons

- ✓ **Source code** availability $\Rightarrow$ full control of the system;
- ✓ Open standards (formats, protocols...) $\Rightarrow$ **interoperability**;
- ✓ **Performances**, **reliability**;
- ✓ **Portability** $\Rightarrow$ variety of supported architectures and hardware;
- ✓ Native **network connectivity**;
- ✓ Scalability $\Rightarrow$ low memory footprint;
- ✓ Diversity and multiplicity of available software.

Economic reasons

- ✓ **Free of charge**, no royalty;
- ✓ Quick and easy implementation;
- ✓ **Independence** against suppliers/providers;
- ✓ Developer community $\Rightarrow$ free and unlimited support, opportunity to speak directly to designers;
- ✓ **Multiplicity of players** $\Rightarrow$ inertia or evolutions of a product are not dictated by a single entity.

Personal reasons

- ✓ Free software and its **4 essential freedoms**:
  - **run** ⇒ no restriction,
  - **study** ⇒ "use the source Luke",
  - **redistribute** ⇒ sell or give the software and its source code,
  - **modify** ⇒ debug, correct or add functionalities.
- ✓ Take part in one of the biggest **community project**;
- ✓ Meet and face the best developers,
- ✓ Expand his resume and adapt to market demands.

In 2012, what are the most important factors in choosing an operating system.

Factor	2012 (N = 1628)
Availability of full source code	41%
Real-time performance	31%
No royalties	31%
Availability of tech support	30%
Freedom to customize or modify	28%
Compatibility w/ other software, systems	27%
Open-source availability	22%
My familiarity with the operating system	21%
The processors it supports	17%
Purchase price	17%
Simplicity / ease of use	16%
Software-development tools available	15%
Small memory footprint	14%
Commercial support	11%
Successful prior use for similar applications	10%
Middleware, drivers, existing code available	10%
Rich selection of services and features	7%
Popularity	6%
The other hardware it supports	6%
The supplier’s reputation	3%

Base: Currently using an operating system

Copyright © 2012 by UBM Electronics. All rights reserved.

Other OS

2012 Embedded Market Study

What operating system or real time operating system (RTOS) was used in your last project? (Please type in your answer -- unaided)

Operating system	2012 (N = 1256)
In house	13 %
Linux	13 %
Wind River VxWorks	6 %
Free RTOS	4 %
Micrium uCOS II	3 %
Custom	2 %
Android	2 %
Freescale MQX	2 %
QNX	2 %
Windows	2 %
Green Hills Integrity	1%
Keil RTX	1%
Texas Instruments DSPBIOS	1%
Express Logic ThreadX	1%
Windows CE	1%
Embedded Linux	1%

Copyright © 2012 by UBM Electronics. All rights reserved.

Licenses

- GNU GPL[1] ⇒ the most widespread, the strictest, most proven and most contagious, it is based on the notion of *copyleft*[2];
- GNU LGPL[3] ⇒ allows dynamic link edition with non-free/proprietary code, widely used for libraries;
- X11/MIT/BSD ⇒ very permissive, make it possible to exploit the code as proprietary;
- *PL ⇒ many publishers have created their own licenses to distribute their open source software (Netscape, IBM, Sun...), the FSF provides a compatibility review of these licenses with GNU's (http://www.gnu.org/licenses/license-list.html) and OSI[4] certify their compliance with OSD[5] (http://www.opensource.org/).

[1] GPL: ***G**eneral **P**ublic **L**icense*
[2] *copyleft*: use of copyright for freedom protection of the software (run, study, redistribute and improve)
[3] LGPL: ***L**esser **G**eneral **P**ublic **L**icense*
[4] OSI: ***O**pen **S**ource **I**nitiative*
[5] OSD: ***O**pen **S**ource **D**efinition*

Limits

- ✗ Not suitable for systems of some tens of kilobytes ("dumb" appliance, sensor, HiFi, remote control...);
- ✗ Source code provided *AS-IS*;
- ✗ No contractual relationship with a supplier/provider;
- ✗ No automatic assistantship (RTFM[1]);
- ✗ Reluctance of developers to deal with changes (inertia principle versus novelty and freedom of choice);
- ✗ Variety of players.

[1] RTFM: ***R**ead **T**he **F*****ing **M**anual*

Why are you not interested in embedded Linux?

Reason	2012 (N = 682)	2011 (N = 802)	2010 (N = 639)	2009 (N = 747)
Incompatible w/ existing software, apps, drivers	39%	41%	41%	38%
Memory usage	25%	26%	20%	21%
Performance or real-time capability	23%	21%	18%	20%
Support	17%	20%	20%	19%
Legal ambiguity	14%	14%	11%	14%
Development tools	14%	14%	14%	12%
Cost (after deployment)	9%	5%	7%	8%
Other*	27%	26%	28%	29%

*Other responses 2012:
- No need = 12%
- Requirements of app, too big/complex = 4%
- Unfamiliarity = 4%

Base = Those who are not considering using Linux

Copyright © 2012 by UBM Electronics. All rights reserved.

Solutions

Types of solutions

Three main approaches are available to Linux embedded developers:

- **"Home made" system**:
  - building/portage of a host cross-platform development environment,
  - choice and inclusion in the toolchain of software building blocks for the system (+ dedicated applications),
  - build and feed target rootfs (filesystem image),
  - automation of previous procedures (scripts, makefiles...) allowing the rapid reconstruction of a binary image containing the kernel and rootfs after modifications.

- **Free distribution (libre)**:
  - ensure that your hardware is supported (architecture, development board...),
  - otherwise adapt the distribution,
  - choose from available softwares those that will appear on the final system.
- **Commercial distribution**:
  - ensure that your hardware is supported (architecture, development board...),
  - purchase one or more licenses of the proprietary developer tools,
  - choose from available softwares those that will appear on the final system.

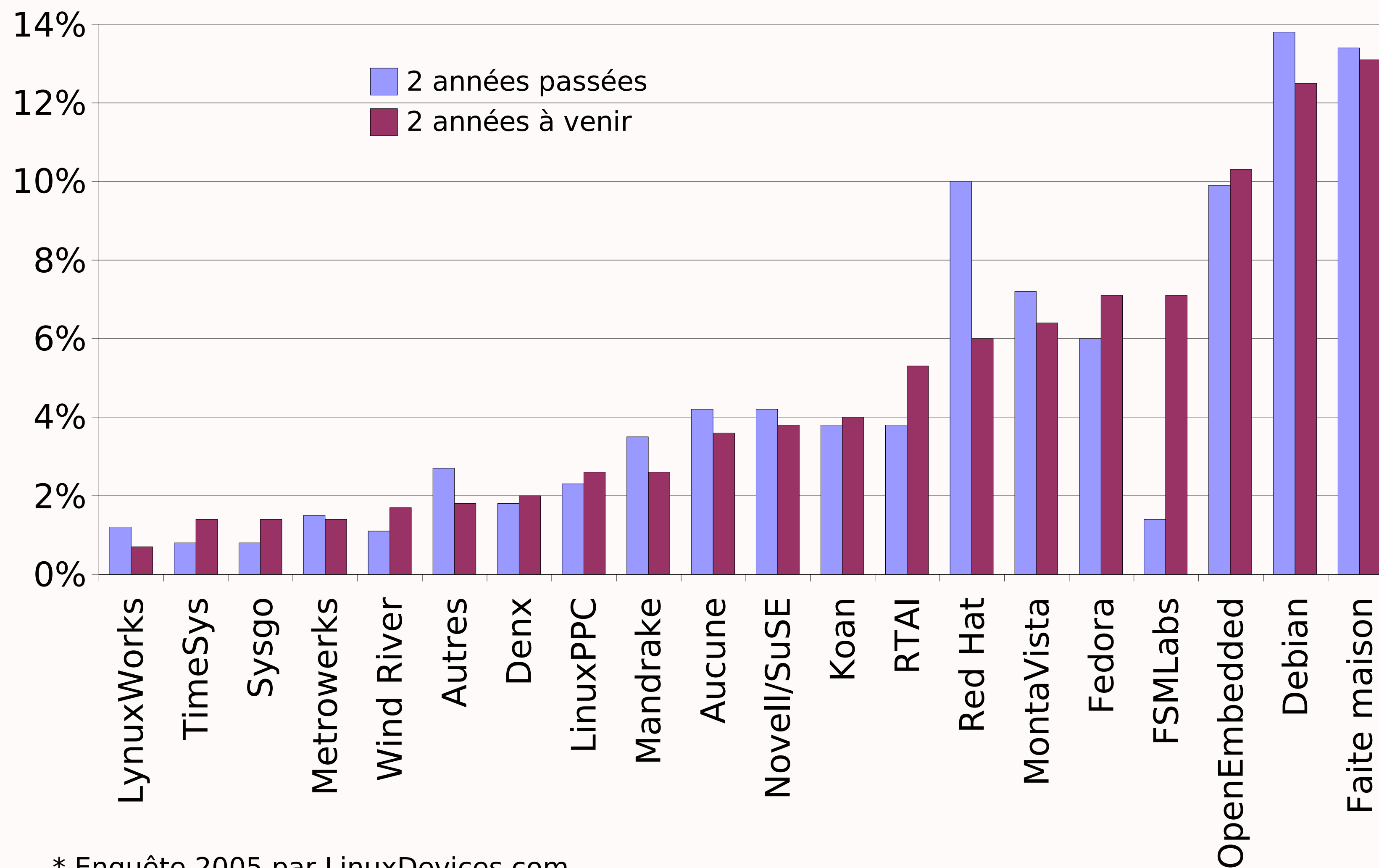
Of course, the three approaches have their advantages and disadvantages depending the profile of the involved developers:

- **"Home made" system**:
  - ✗ longer to set up,
  - ✗ ⇒✓requires deeper knowledge of tools and mechanisms involved, which can be annoying at first but very useful later,
  - ✗ no integrated IDE[1],
  - ✓ complete control over the system and its components,
  - ✓ total independence.

---

[1] IDE: ***I**negrated **D**evelopment **E**nvironment*

- **Free distribution (libre)**:
  - ✓ quick start,
  - ✓ significant number of users and contributors,
  - ✗ full control possible, but less obvious,
  - ✗ IDE is not always available,
  - ✓ total independence.
- **Commercial distribution**:
  - ✓ quick start,
  - ✓ full-featured IDE,
  - ✓ dedicated help (fees often apply),
  - ✗ cost of the development toolchain,
  - ✗ less control,
  - ✗ near reliance on provider and its toolchain.


Fournisseurs/Distributions Linux préférés dans l'embarqué*
14%
12%
10%
8%
6%
4%
2%
0%
2 années passées
2 années à venir
LynuxWorks
TimeSys
Sysgo
Metrowerks
Wind River
Autres
Denx
LinuxPPC
Mandrake
Aucune
Novell/SuSE
Koan
RTAI
Red Hat
MontaVista
Fedora
FSMLabs
OpenEmbedded
Debian
Faite maison
* Enquête 2005 par LinuxDevices.com

Product oriented platforms

- Middleware geared toward a market segment (eg, smartphones, tablets, routers, IVI[1]...);
- SDK[2] with high level language:
  - facilitate and accelerate developments,
  - federate developers.
- Simplifying access to field resources;
- Free, open source or commercial;
- Related or not to a hardware platform.

[1]IVI: ***I**n-**V**ehicle **I**nfotainment*
[2]SDK: ***S**oftware **D**evelopment **K**it*

Base software components

Linux kernel

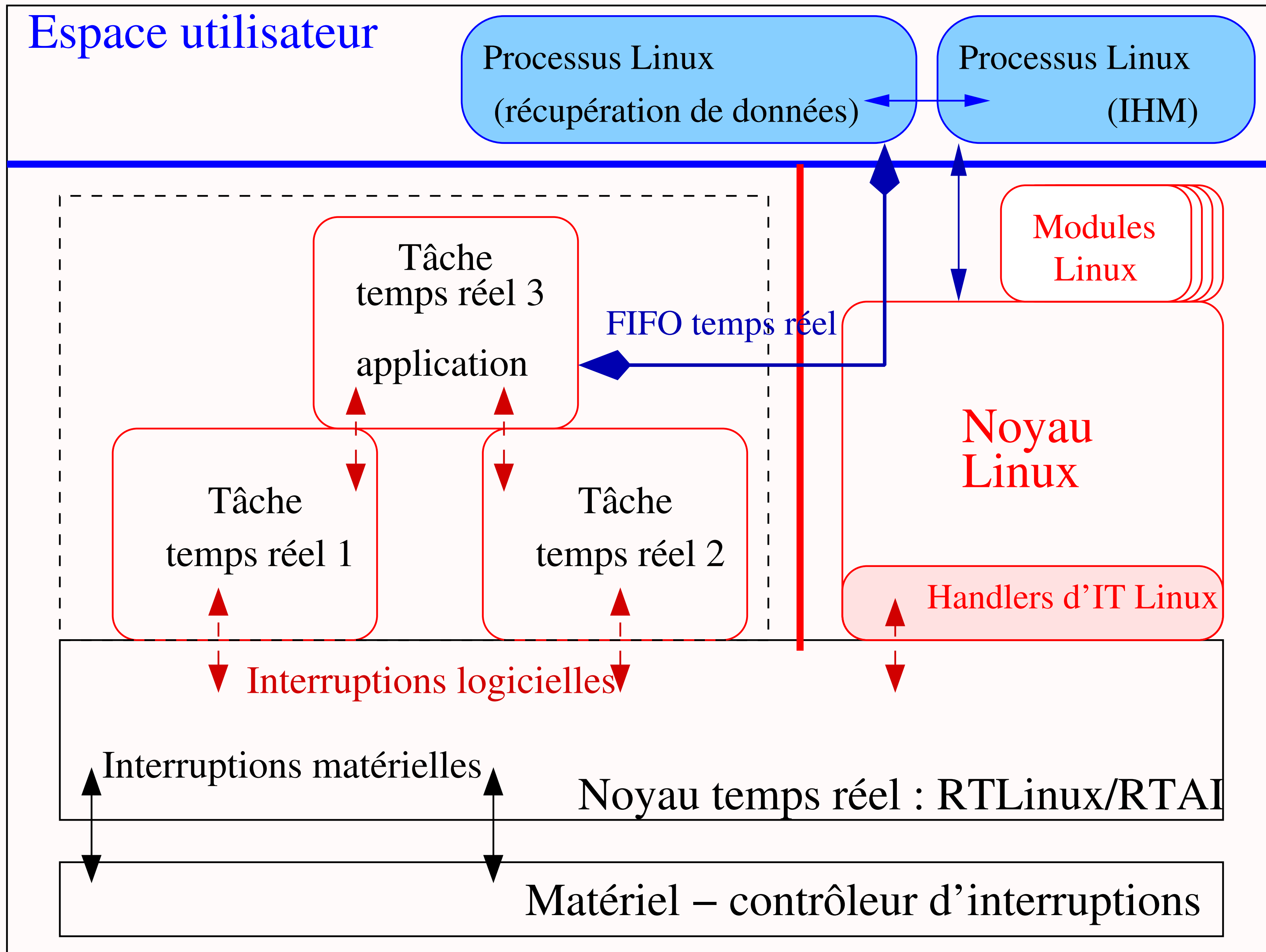
- **Linux *vanilla*** (http://www.kernel.org/) $\Rightarrow$ the regular Linux kernel;
- μ**Clinux** (http://www.uclinux.org/) $\Rightarrow$ kernel patch to support MMU[1]-less architectures and some specific hardware relative to it (integrated in regular kernel from 2.6).

[1] MMU: ***M**emory **M**anagement **U**nit*

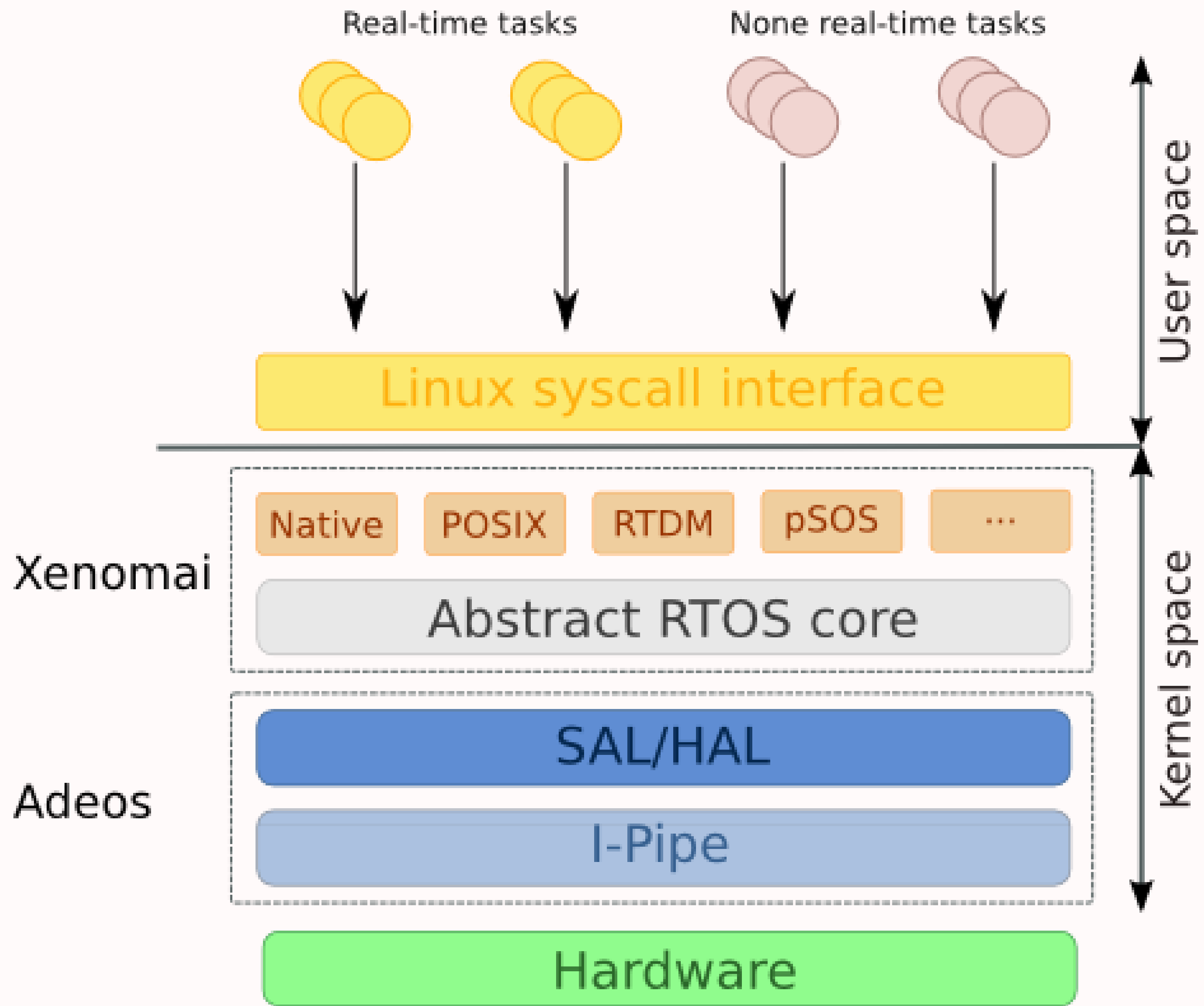
Real-time extensions

- **Open RTLinux/Wind River Real-Time Core** (http://fr.wikipedia.org/wiki/RTLinux) ⇒ pioneered the patented, microkernel-based, technique of hard real-time on Linux, but the free version is much less advanced than the commercial version;
- **RTAI[1]/Linux** (http://www.rtai.org/) ⇒ POSIX[2] API and free real-time microkernel allowing the coexistence of hard real-time tasks with Linux kernel as a lower priority task (now based on patent-free approach ADEOS);
- **Xenomai** (http://www.xenomai.org) ⇒ free successor of RTAI whose programming API is based on the concept of *skins* to allow maximum reuse of existing code (POSIX, proprietary OS, etc), and scheduling based on a microkernel approach (Nucleus) or ***PREEMPT_RT***;

[1] RTAI: ***R**eal **T**ime **A**pplication **I**nterface*
[2] POSIX: ***P**ortable **O**perating **S**ystem **I**nterface*



Source: Nicolas Ferre (http://noglitch.free.fr/)



Source: Xenomai.org

- ***low latency***, ***O(1) scheduler***, ***preemtible kernel***... ⇒ kernel patches for improved system calls latency, better scheduler responsiveness or kernel preemption, these are often developed and sponsored by commercial distributions (RedHat, MontaVista, TimeSys...);
- ***PREEMPT_RT*** ⇒ patch to bring native preemption and support of hard real-time constraints to the Linux kernel (maintained by the community of embedded developers).

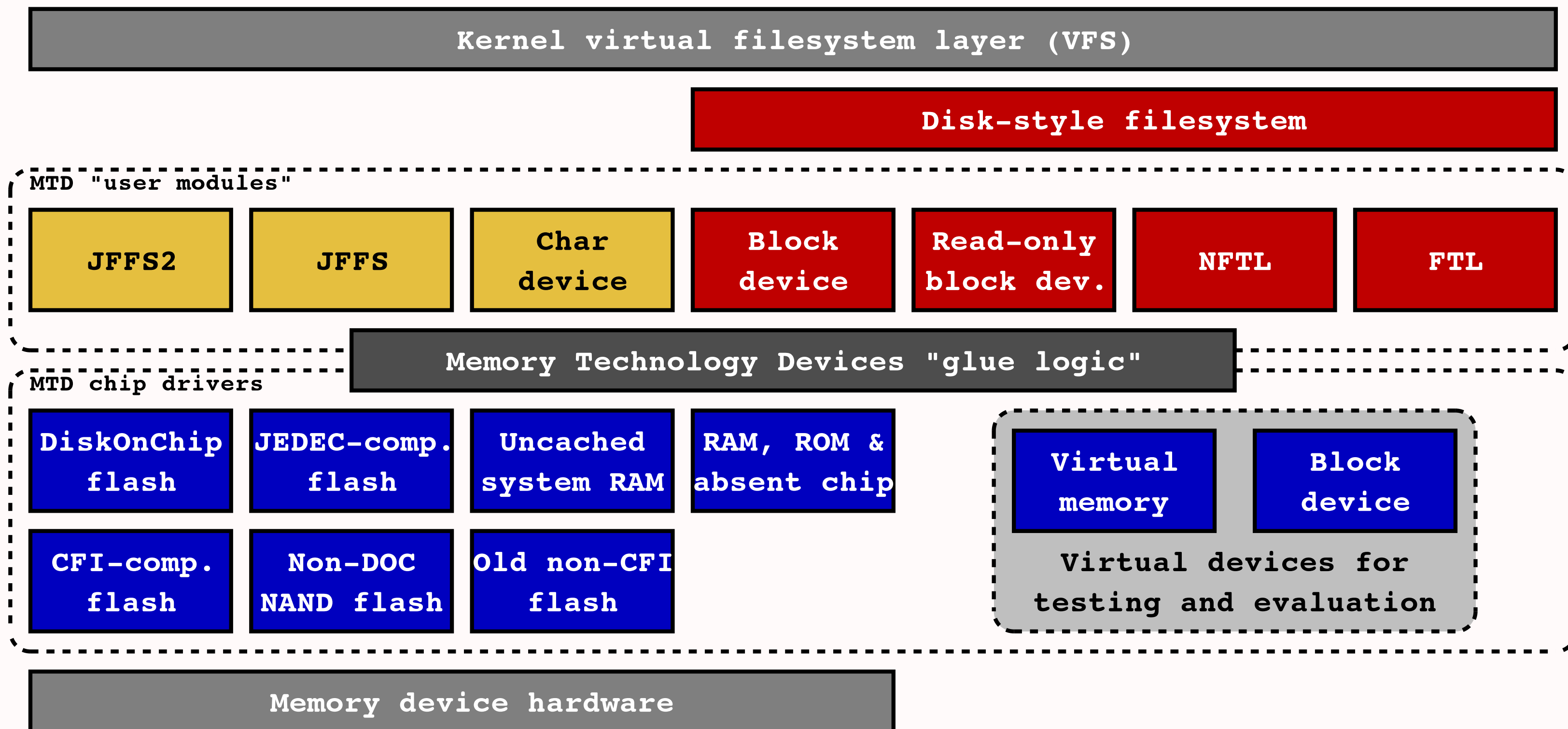
Filesystem for embedded systems

- **YAFFS2** (http://www.yaffs.net/) ⇒ robust filesystem (journaling, error correction) for NAND flash[1];
- **JFFS2** (http://sources.redhat.com/jffs2/) ⇒ compressed filesystem for flash memory, crash and powerfail resistant, taking into account the specificities of memory storage media via the Linux MTD[2] layer (see also UBIFS);
- **ROMFS** (http://romfs.sf.net/)/**CRAMFS**[3] (http://sf.net/projects/cramfs/) / **SquashFS** (http://squashfs.sourceforge.net/) ⇒ read-only filesystems providing static storage (built on the development system and stored on ROM, flash or RAM) with minimal features and size (no permission, no modification date, compression for CRAMFS et SquashFS...);

[1] NAND/NOR flash: page and bloc access (∼ hard drive) / random access (∼ RAM)
[2] MTD: *__M__emory __T__echnology __D__evices*
[3] CRAMFS: *Compressed __ROM__ __F__ile__S__ystem*

- **LogFS** (http://logfs.org) ⇒ log-structured filesystem, efficient even on large Flash storage (see also NILFS);
- **AuFS** (http://aufs.sourceforge.net/) / **OverlayFS** ⇒ overlay filesystem and service that combine two or more filesystems in the same tree (usually a RW over a RO filesystem);
- **TMPFS/RAMFS** ⇒ RAM filesystems with swap (TMPFS only, requires a MMU) and dynamically resizable, these are often used to store non persistent datas (/tmp, /var, logs...);
- **PRAMFS**[1] (http://pramfs.sourceforge.net/) ⇒ filesystem for non volatile RAM (battery powered), persistant over reboot.

[1] PRAMFS: ***P**rotected and **P**ersistent **RAM** **F**ile**S**ystem*



Source: Karim Yaghmour - *Building Embedded Linux Systems* (http://www.embeddedtux.org/)

Conventional filesystems

- **Ext2** (http://e2fsprogs.sf.net/ext2.html) ⇒ was the default Linux filesystem, can be mounted with synchronous writes (`mount -o sync`) to ensure data integrity at the expense of performance;
- Journaled (**Ext3/4**, **ReiserFS**, **XFS**, **JFS**...) ⇒ keeps track of the changes in a journal (transactions) before committing them to the main filesystem, quicker to bring back online and less likely to become corrupted in case of power failure;
- *Copy-on-write* (**BtrFS**, **ZFS**) ⇒ modern filesystems with pooling, snapshots, checksums, integral multi-device spanning...
- **MS-DOS FAT 12/16/32** ⇒ was the default filesystem of the Microsoft OS, different versions accomodate to size of storage (still widely used in CE[1]).

[1]CE: *Consumer Electronics*

C library (libc)

- **Glibc** (http://www.gnu.org/software/libc/libc.html) / **EGlibc** (http://www.eglibc.org) $\Rightarrow$ official C library on Linux systems, full-featured, powerful, multi-platform and very well documented but still bulky and not very suitable in small footprint systems ($\Rightarrow$ EGlibc);
- μ**Clibc** (http://www.uclibc.org/) $\Rightarrow$ almost fully compatible (source code) with the Glibc, it is much more suitable for embedded systems because designed to be as small as possible, it also supports kernel μClinux (systems without MMU);
- **diet libc** (http://www.dietlibc.org/) $\Rightarrow$ lightweight C library for creating statically linked binaries (not compatible with Glibc);
- **Newlib** (http://www.sourceware.org/newlib/) $\Rightarrow$ association of several embedded libraries that can be used without any OS (BSP[1]).

[1]BSP: ***B**oard **S**upport **P**ackage*

Basic tools

- **BusyBox** (http://www.busybox.net/) $\Rightarrow$ Swiss army knife for embedded Linux, it reimplements over 200 major utilities available on Linux systems in one lightweight executable (shells, console-tools, procps, util-linux, modutils, NetUtils ...);
- **TinyLogin** (http://tinylogin.busybox.net/) $\Rightarrow$ perfect complement to BusyBox (today integrated) for embedded systems using authentication (access control and management of users, groups, passwords, rights...);
- **EmbUtils** (http://www.fefe.de/embutils/) $\Rightarrow$ set of common Unix tools optimized for size and based on diet libc;
- **Outils GNU** (http://www.gnu.org/directory/GNU/) $\Rightarrow$ standard tools from the GNU project.

Network servers

- Web:
  - **Boa** (http://www.boa.org/),
  - **BusyBox::httpd** (http://www.busybox.net/),
  - **LightTPD** (http://www.lighttpd.net/),
  - **thttpd** (http://www.acme.com/software/thttpd/),
  - **Mbedthis AppWeb** (http://www.mbedthis.com/),
  - **GoAhead WebServer** (http://www.goahead.com/products/web_server.htm),
  - **Apache** (http://www.apache.org/)...
- FTP:
  - **sftpd** (http://safetp.cs.berkeley.edu/),
  - **ProFtpd** (http://proftpd.linux.co.uk/),
  - **tftpd**...

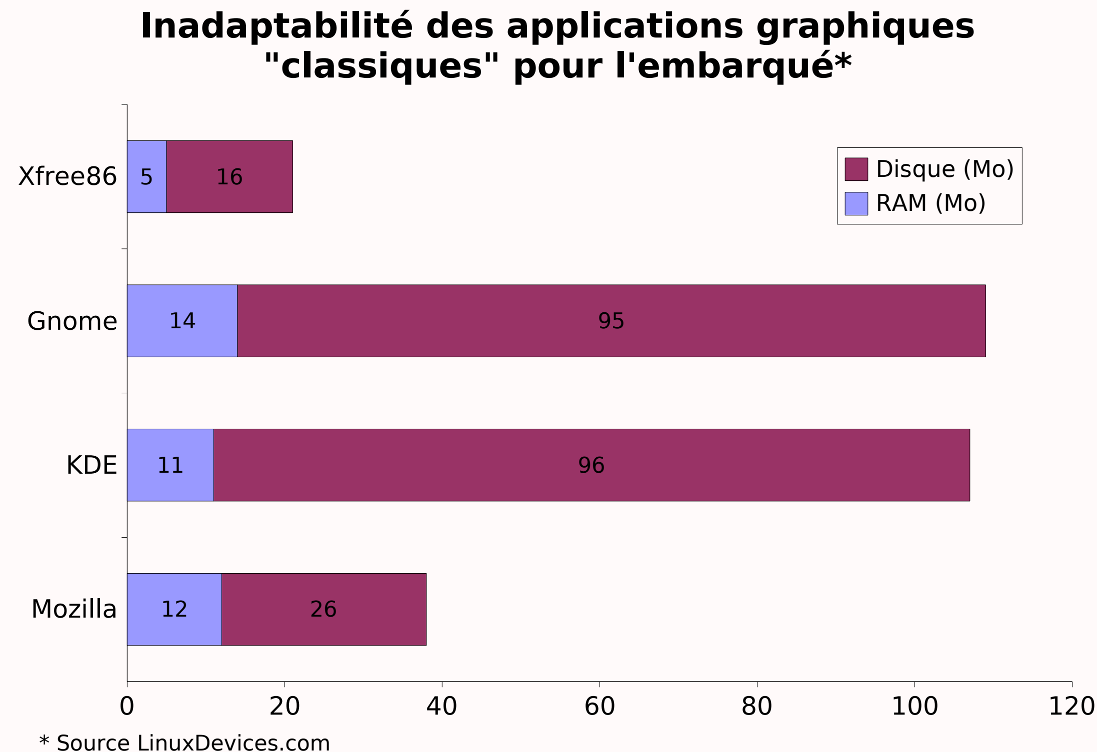
- Remote access:
  - **OpenSSH** (http://www.openssh.com/),
  - **telnetd**,
  - **utelnetd**,
  - **gettyd**,
  - **pppd**...
- DHCP:
  - **BusyBox::udhcp** (http://www.busybox.net/),
  - **dhcpd**...
- Autres:
  - **Zebra** (http://www.zebra.org/),
  - **snmpd**...

Databases

- **Berkeley DB** (http://www.sleepycat.com/) ⇒ open source, GPL compliant, commercial, non-relational, simple, less than 500 kB, multi-platform;
- **MySQL** (http://www.mysql.com/) ⇒ open source (GPL or commercial), relational, SQL, transactional, fast, multi-platform, widespread;
- **SQLite** (http://www.sqlite.org/) ⇒ open source, relational, SQL, transactional, server-less (standalone library), multi-platform, single file, widespread in embedded systems, less than 275 kB;
- **DB2 Everyplace** (http://www.ibm.com/software/data/db2/everyplace/) ⇒ commercial, less than 200 kB, multi-platform.

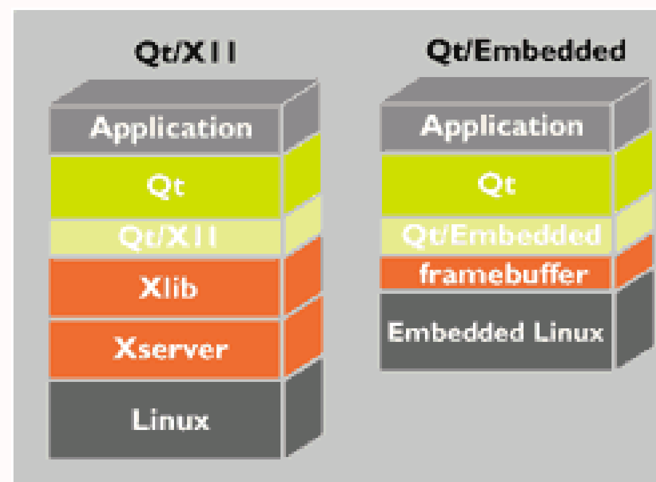
GUI[1]

- Regular graphical tools and libraries used on desktop computers are not suitable for embedded systems due to their size on storage and memory.



[1]GUI: ***G**raphical **U**ser **I**nterface*

- **DirectFB** (http://www.directfb.org/) ⇒ overlay of the Linux *framebuffer* for building fast and light GUI with full hardware abstraction;
- **Gtk+** (http://www.gtk.org/) ⇒ a version of that library (very common in the Unix world) uses DirectFB to avoid the use of a X Window server;
- **Qt/Embedded** (http://www.trolltech.com/products/embedded/) ⇒ layer allowing the use of the Qt library above the Linux *framebuffer* (from 800 kB à 3 MB);

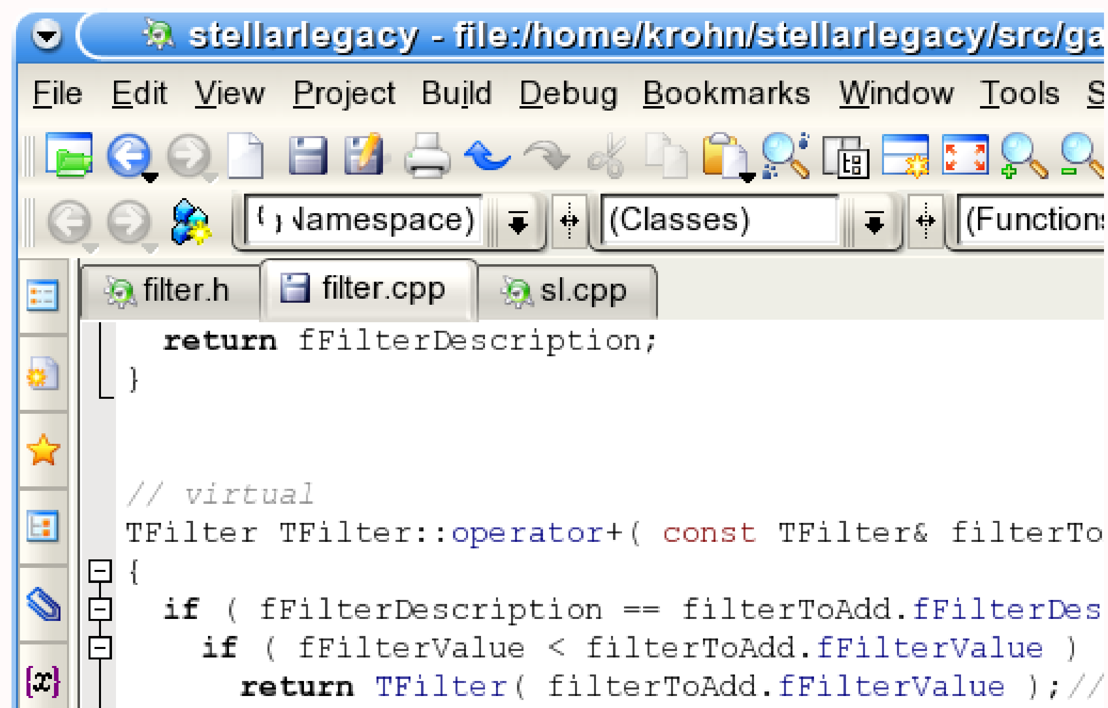


- **MicroWindows/Nano-X** (http://www.microwindows.org/)
  ⇒ multi-platform graphical environment with its own API, but also a library compatible with X Window (100 kB);
- **X Window System (XFree86/X.org)** ⇒ (http://www.xfree86.org/ / http://www.x.org/) open source implementations of the historical graphical server of all Unix systems, are fast and optimized for many chipsets but memory and disk space intensive;
- **Tiny-X** (http://www.xfree86.org/) ⇒ implementation of a X Window server for embedded systems (1 MB);
- **SDL**[1] (http://www.libsdl.org/) ⇒ multi-platform library for the development of multimedia graphical applications.

[1] SDL: *Simple DirectMedia Layer*

IDE[1]

- Eclipse (http://www.eclipse.org/);
- KDevelop (http://www.kdevelop.org/);
- Vim (http://www.vim.org/);
- Emacs (http://www.emacs.org/).

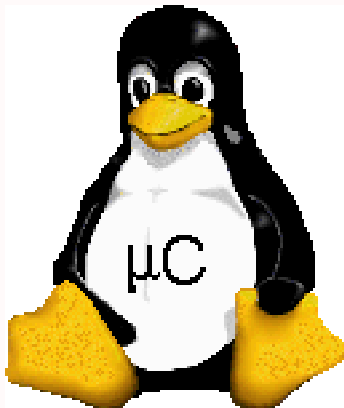


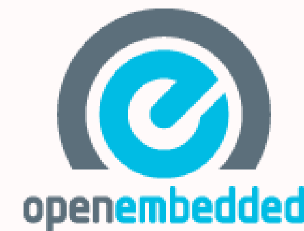
[1]IDE: ***I****ntergated* ***D****evelopment* ***E****nvironment*

References

Free distributions

- μClinux-dist (http://www.uclinux.org/pub/uClinux/dist/);
- SnapGear Embedded Linux (http://www.snapgear.org/);
- Yocto Project / OpenEmbedded;
- Pengutronix PTXdist (http://www.pengutronix.de/software/ptxdist_en.html);
- Denx ELDK[1] (http://www.denx.de/ELDK/).



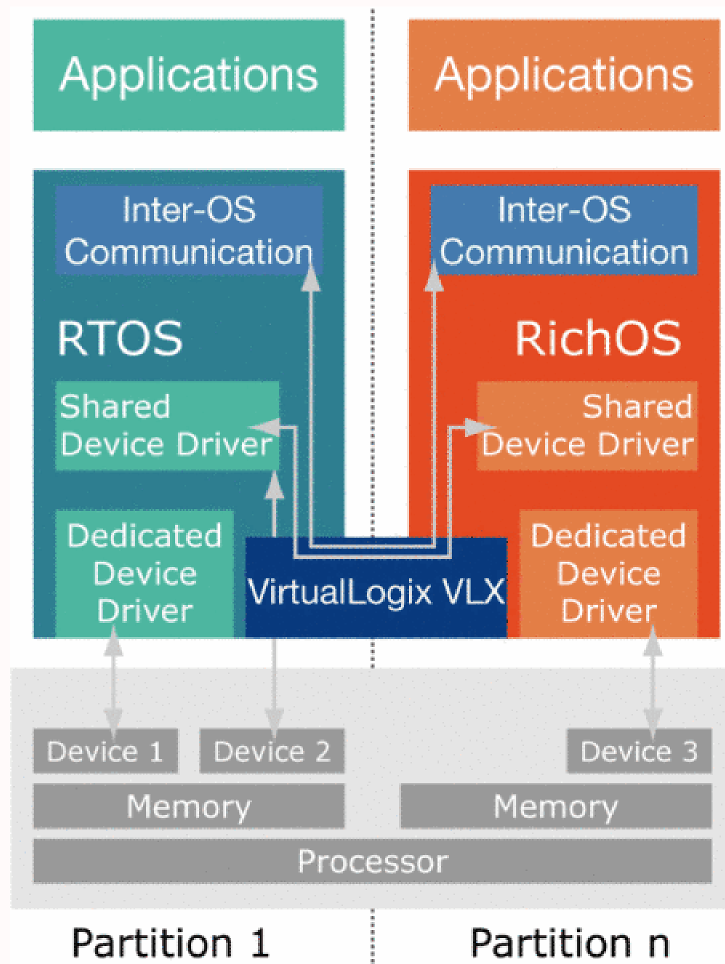


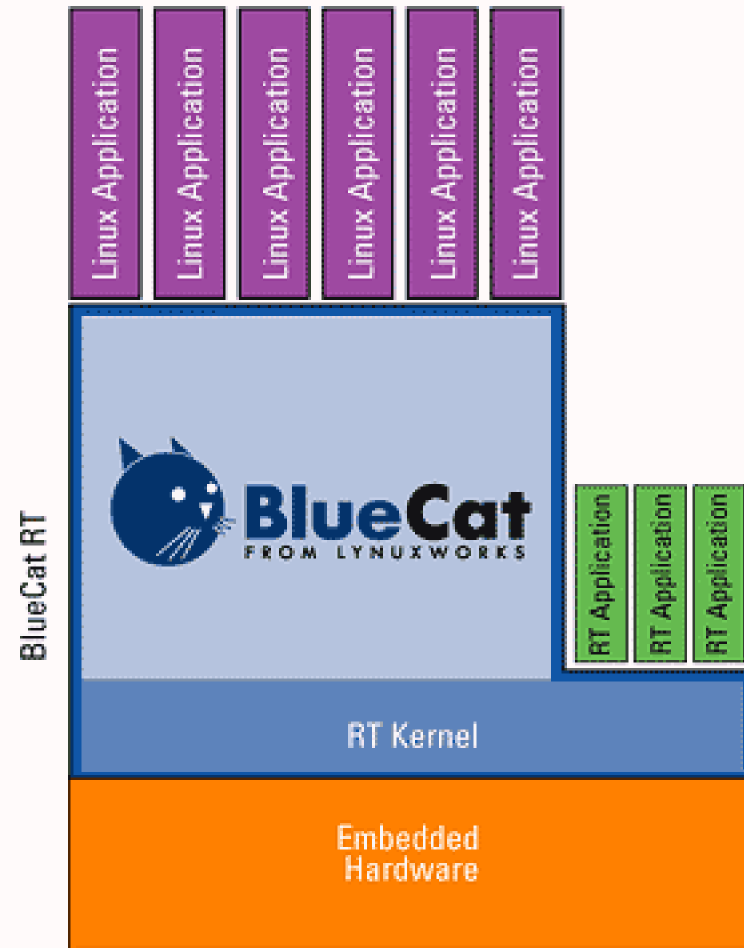
[1] ELDK: ***E**mbedded **L**inux **D**evelopment **K**it*

Commercial distributions

- FSMLabs (http://www.rtlinux.com/);
- VirtualLogix (http://www.virtuallogix.com/);
- Koan (http://www.klinux.org/);
- LynuxWorks (http://www.lynuxworks.com/);
- Intel/Windriver (http://www.windriver.com/);
- MontaVista (http://www.mvista.com/);
- SysGo (http://www.elinos.com/);
- TimeSys (http://www.timesys.com/).

WIND RIVER


Applications
Applications
Inter-OS Communication
Inter-OS Communication
RTOS
RichOS
Shared Device Driver
Shared Device Driver
Dedicated Device Driver
VirtualLogix VLX
Dedicated Device Driver
Device 1
Device 2
Device 3
Memory
Memory
Processor
Partition 1
Partition n


Linux Application
Linux Application
Linux Application
Linux Application
Linux Application
Linux Application
BlueCat RT
BlueCat
FROM LYNUXWORKS
RT Application
RT Application
RT Application
RT Kernel
Embedded Hardware

Product oriented platforms

- Android (http://developer.android.com/);
- Tizen (http://developer.tizen.org/);
- Bada (http://developer.bada.com/);
- WebOS (http://developer.palm.com/);
- Zeroshell (http://www.zeroshell.net/).

Essentials

Unix concepts and orthodoxy

"UNIX is basically a simple operating system, but you have to be a genius to understand the simplicity." - Dennis Ritchie

- Everything is file (datas, drivers, links, pipes, sockets);
- Modularity and pipes "|":
  - simple programs that do one thing and do it well,
  - programs that work together and combine to make complex things,
  - focus on text stream as a universal interface,
  - eg, `du -ks * | sort -n`.

Linux boot process analysis

- **Firmware (bootstrap)** ⇒ placed in a ROM/Flash to the first address accessed by the processor after a reset, it initializes the CPU and passes control to the bootloader;
- **Bootloader** ⇒ responsible for launching the kernel by running it in place (XIP[1]) or placing it in RAM after downloading from:
  - a predetermined address on a storage medium (ROM, Flash, hard drive, CDROM...),
  - a filesystem it knows how to access,
  - the network (BOOTP/TFTP[2]).

[1] XIP: *eXecute **I**n **P**lace*
[2] TFTP: ***T**rivial **F**ile **T**ransfert **P**rotocol*

- **kernel** ⇒ after an initialization stage of all its components, it mounts the root filesystem (rootfs) available on:
  - a storage medium, or
  - RAM, preloaded by the bootloader, or
  - the network (NFS[1]).

  before the startup of the first process (`init`);
- The **`init`** process launches applications and other system services...

[1]NFS: ***N**etwork **F**ile**S**ystem*

Compilation process

```
gcc -v helloworld.c -o helloworld
```

- **Preprocessor** (CPP[1]) ⇒ handles macro commands from C files (`#include`, `#define`, `#ifdef`, `__FUNCTION__`...);
- **Compiler** (CC[2]) ⇒ transforms the C source files in assembly source files dedicated to a platform;
- **Assembler** (AS[3]) ⇒ transforms assembly source files in binary objets (BFD[4] library);
- **Linker** (LD[5]) ⇒ produces an executable from the binary objects and static libraries (archives).

---

[1] CPP: ***C*** ***Pre****P**rocessor*

[2] CC: ***C*** ***C****ompiler*

[3] AS: ***AS****sembler*

[4] BFD: ***B****inary* ***F****ile* ***D****escriptor*

[5] LD: *Link e****D****itor*

Binary link edition

Static links

```
gcc -Wall -O2 -o libmisc.o -c libmisc.c
ar -rc libmisc.a libmisc.o
gcc -Wall -O2 -o app.o -c app.c
gcc -L. -static -o app_static app.o -lmisc
```

- All dependencies are resolved during link edition;
- The resulting executable is much larger because it contains portions of the code from libraries it uses;
- It will run whatever versions of shared libraries are present on the target (it does not use it).

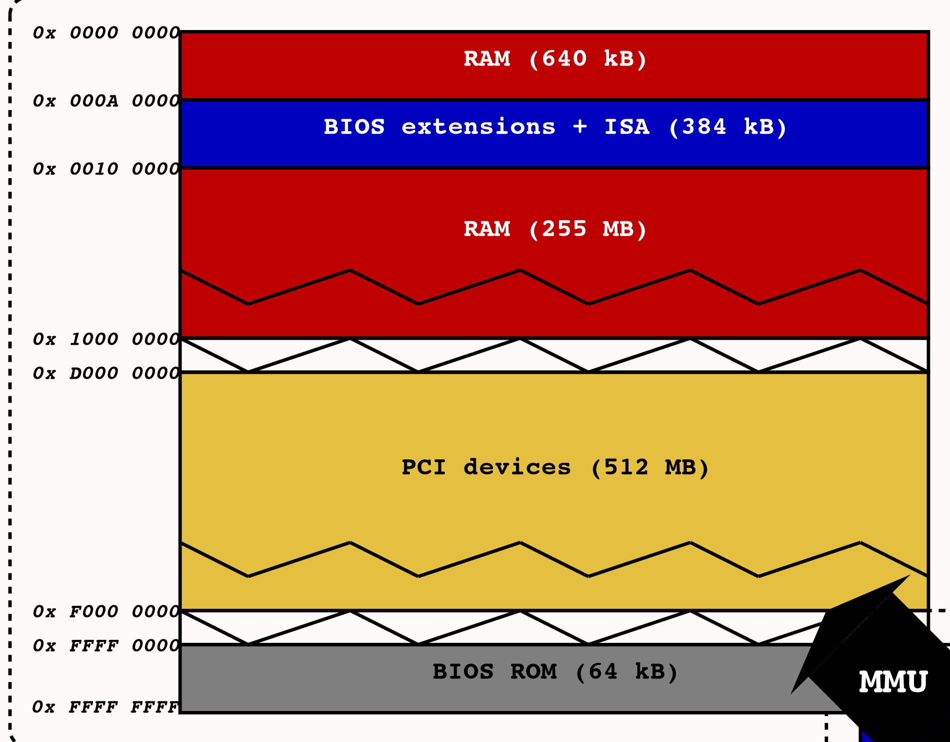
Dynamic links

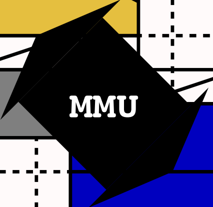
```
gcc -Wall -O2 -fpic -o libmisc.po -c
libmisc.c
gcc -shared -o libmisc.so libmisc.po
gcc -Wall -O2 -o app.o -c app.c
gcc -L. -o app_dynamic app.o -lmisc
```

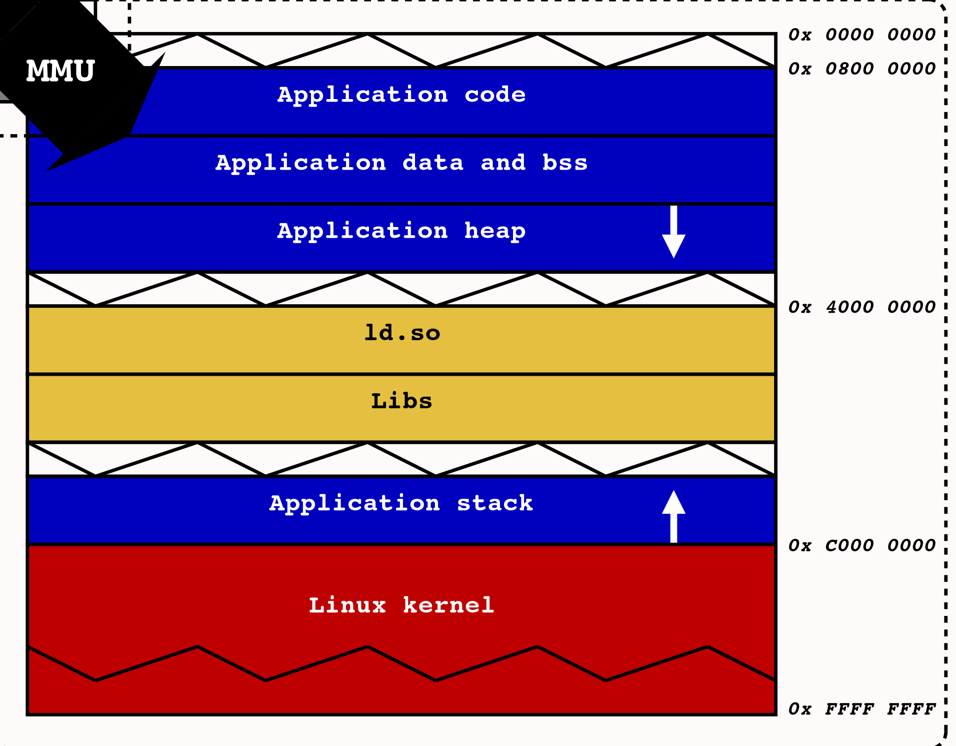
- Default type of link edition on platforms that support this mechanism;
- The final link editing is achieved when loading the executable;
- If N executable use the same version of a shared library, it is loaded only once in memory;
- The executable is smaller containing only its own code.

Process of loading an executable dynamically linked

- When the process is created, the kernel loads the executable and the dynamic loader (`ld-linux.so` for ELF binaries) into memory using `mmap()` system call;
- Control is given to the dynamic loader;
- The charger inspects the executable and libraries available on the system (via `ld.so.cache` and `ld.so.conf`) to resolve dependencies (data and functions) and find the necessary libraries;
- It then loads into memory all the necessary libraries to predefined addresses in the virtual memory space of the process;
- The charger finally jumps to the start point of the program and this one can then start.


PC-x86 simplified physical memory map
0x 0000 0000
RAM (640 kB)
0x 000A 0000
BIOS extensions + ISA (384 kB)
0x 0010 0000
RAM (255 MB)
0x 1000 0000
0x D000 0000
PCI devices (512 MB)
0x F000 0000
0x FFFF 0000
BIOS ROM (64 kB)
0x FFFF FFFF


MMU


Linux process virtual memory map
0x 0000 0000
0x 0800 0000
Application code
Application data and bss
Application heap
0x 4000 0000
ld.so
Libs
Application stack
0x C000 0000
Linux kernel
0x FFFF FFFF

Tools related to shared libraries

- `ldd` $\Rightarrow$ display the shared library dependencies of a dynamically linked executable or another shared library;
- `ldconfig` $\Rightarrow$ create the symbolic links and cache file `ld.so.cache` used by the dynamic loader according to `/lib`, `/usr/lib` and other directories listed in `ld.so.conf`;
- `ltrace` $\Rightarrow$ catch and print the calls to shared libraries for one process.

Static vs dynamic

- Static if:
  - dynamic is not supported (often the case with MMU-less platforms),
  - few executables share the same libraries,
  - only few functions of each library are used.
- Dynamic if:
  - memory resources available are very limited,
  - many applications on the target,
  - need to upgrade or correct the libraries without updating the entire target.

Executables

Common formats

- ELF[1] ⇒ binary format for executables, objects and libraries, it is the standard for most Unixes (including Linux);
- a.out[2] ⇒ the default output format of the assembler and the link editor of the Unix systems;
- bFLT[3] (Flat) ⇒ lightweight file format for executables, derived from `a.out` format used by the project μClinux, supports compression;
- COFF[4] ⇒ binary format from the Unix System V ABI[5], it is the ancestor of ELF.

[1] ELF: ***E****xecutable and* ***L****inkable* ***F****ormat*
[2] a.out: ***a****ssembler* ***out****put*
[3] bFLT: ***b****inary* ***FL****a****T*** *format*
[4] COFF: ***C****ommon* ***O****bject* ***F****ile* ***F****ormat*
[5] ABI: ***A****pplication* ***B****inary* ***I****nterface*

Operations on executables

- **lightening** $\Rightarrow$ utility `strip` removes symbols, debug informations and other unnecessary contents from a binary file (executable or library);
- **conversion** $\Rightarrow$ utility `elf2flt` converts an ELF binary to bFLT;
- **compression** $\Rightarrow$ bFLT format supports compression (full or only datas) with runtime decompression by the kernel (`elf2flt [-z|-d]`).

μClinux vs Linux

Main differences

- μClinux is tailored to platforms without MMU
  - no memory protection,
  - no virtual memory (flat memory model).

Consequences

- `fork()` system call is not implemented $\Rightarrow$ use of `vfork()` (API BSD):
  - father and son share the same memory space (including stack), and
  - the father is suspended until his son calls `execve()` or `exit()`.
- Fast fragmentation if many dynamic memory allocations or releases (`malloc()/free()`) $\Rightarrow$ prefer allocating a pool at application startup;

- Fixed size stack (at compile time);
- Use of relocatable binaries:
  - relative addressing (PIC[1]) $\Rightarrow$ binary limited to 32 kB (16 bits jump of 68k), or
  - absolute addressing fully relocatable (references modified at loading time) $\Rightarrow$ heavier and slower to load.
- No swap mecanism.

[1] PIC: ***P**osition **I**ndependant **C**ode*

Methods and development tools

Terminology

- We distinguish two entities:
  - the **target** is the hardware platform (device) that will run the embedded OS and applications,
  - the **host** is the development platform (desktop) on which the software for the target is prepared.
- Host and target hardly share the same hardware architecture and sometimes neither the same OS;
- A single host can be used to develop many different targets at the same time.

Development method

- Usually, we distinguish 4 development methodologies for embedded systems:
  - online development,
  - development through removable storage,
  - on target development,
  - development with prototype.
- These methodologies are more or less dictated by storage constraints, performance and accessibility of the target system.

Online development

- The target is connected to the host by a physical link (Ethernet, USB, serial, JTAG...);
- The link is used to:
  - remotely update the target, a/o
  - debug the target, a/o
  - let the target download its kernel and rootfs (TFTP, NFS...).
- It is the most encountered configuration.

Development through removable storage

- The target has a minimal bootloader;
- Developer puts the kernel and rootfs on the removable storage (CompactFlash, EEPROM...) using a programer on the host;
- The media storage is then set up on the target;
- ROM emulator improves the process to make it look like a online development.

On target development

- Only possible on embedded systems with sufficient storage space and memory to run a compiler (eg SOB systems with hard drive);
- The target has its own native development toolchain (editor, compiler, debugger ...);
- The developer accesses the target either directly using a keyboard and a screen, or through the network from the host (ssh, telnet...);
- Generally based on a Linux distribution, it may possibly be "lightened" in the transition to the final system (suppression of design tools, documentation, unnecessary packages...).

Development with prototype

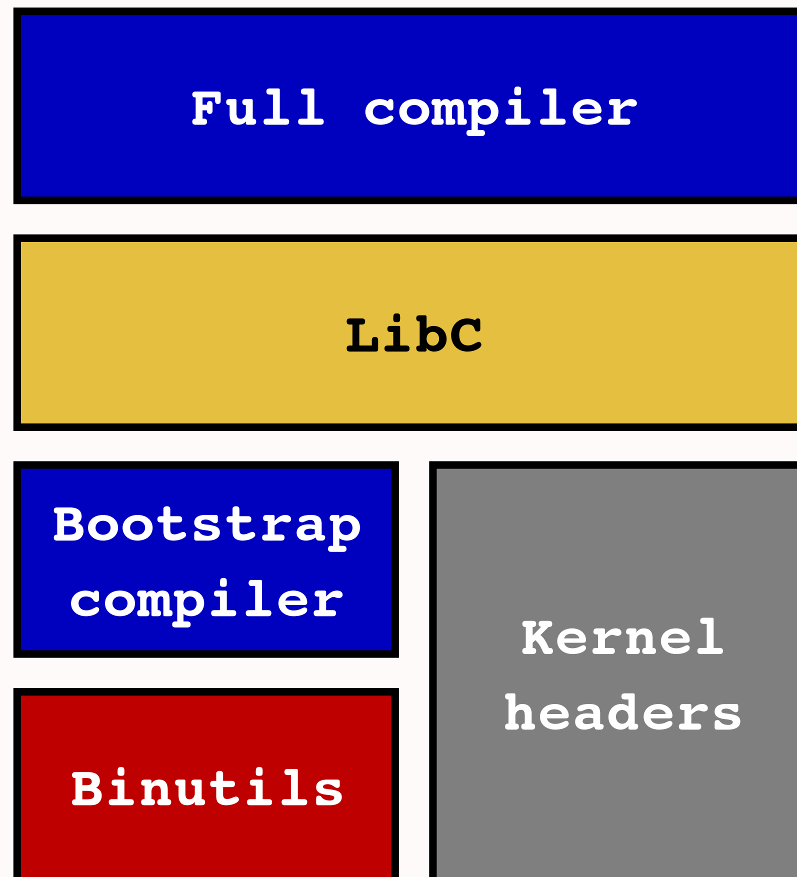
- Development is made on target or on a development platform (similar to the target) from a Linux distribution on hard drive;
- Development (in parallel?) of a light rootfs that:
  - contains only the directory tree, utilities and libraries (beware versions) required for applications,
  - will be later placed on the target (eg CompactFlash to overcome a hard drive) after integration of applications.

Cross-compilation

Compilation toolchain

- We talk about a *Cross-Platform Development Toolchain*;
- It consists of the following:
  - a set of binary file manipulation tools (binutils),
  - a C/C++ compiler (GCC),
  - a kernel (Linux),
  - a C library (μClibc).

Interdependencies in the toolchain



Making of the cross-compilation toolchain

- The host must already have a local toolchain (see distributions) enabling it to compile native applications (for itself);
- GNU target naming:
  - ARM ⇒ `arm-linux`
  - PowerPC ⇒ `powerpc-linux`
  - MIPS (big endian) ⇒ `mips-linux`
  - MIPS (little endian) ⇒ `mipsel-linux`
  - i386 ⇒ `i386-linux`...

- Target and pathes setup:

```
$ export TARGET=arm-linux
$ export PREFIX=/usr/local
$ export INCLUDE=$PREFIX/$TARGET/include
```

- Binutils configuration:

```
$ tar zxvf binutils-2.10.1.tar.gz
$ mkdir build-binutils
$ cd build-binutils
$ ../binutils-2.10.1/configure
  -target=$TARGET -prefix=$PREFIX
$ make
$ make install
```

The `$PREFIX/bin` directory contains `arm-linux-ar`, `arm-linux-as`, `arm-linux-ld`, `arm-linux-nm`, `arm-linux-objdump`, `arm-linux-strip`...

- Installation of the *bootstrap cross-compiler*:

```
$ tar zxvf gcc-2.95.3.tar.gz
$ mkdir build-bootstrap-gcc
$ cd build-bootstrap-gcc
$ ../gcc-2.95.3/configure -target=$TARGET
  -prefix=$PREFIX -without-headers
  -with-newlib -enable-languages=c
$ make all-gcc
$ make install-gcc
```

- Linux kernel headers setup:

```
$ tar jxvf linux-2.4.24.tar.bz2
$ cd linux-2.4.24
$ make ARCH=arm CROSS_COMPILE=$TARGET-
  menuconfig
$ mkdir $INCLUDE
$ cp -r linux/ $INCLUDE
$ cp -r asm-generic/ $INCLUDE
$ cp -r asm-arm/ $INCLUDE/asm
```

- Installation of the C library:

```
$ tar jxvf uClibc-0.9.16.tar.bz2
$ cd uClibc-0.9.16
$ make CROSS=$TARGET- menuconfig
$ make CROSS=$TARGET-
$ make CROSS=$TARGET- PREFIX="" install
```

- Full cross-compiler setup:

```
$ mkdir build-gcc
$ cd build-gcc
$ ../gcc-2.95.3/configure -target=$TARGET
  -prefix=$PREFIX -enable-languages=c,c++
$ make all
$ make install
```

How to use the cross-compilation toolchain

```
$ arm-linux-gcc exemple.c -o exemple
$ arm-linux-size exemple
$ arm-linux-strip exemple
```

...

ScratchBox/Crosstool-NG/buildroot

- `www.scratchbox.org`, `crosstool-ng.org`, `buildroot.uclibc.org`;
- Toolkits simplifying the creation of complete cross-compilation toolchains;
- Various features:
  - management of cross-compilation and cross-configuration,
  - sandbox mechanism (QEMU + chroot) to isolate the host from target,
  - kernel and rootfs generation for *buildroot*.

Yocto Project/OpenEmbedded

- `www.yoctoproject.org`, `www.openembedded.org`;
- Frameworks for creating your own embedded distribution;
- Based on a common framework *OpenEmbedded-Core* and the *BitBake* tool;
- Manage cross-compilation;
- Support several package managers;
- Tests with QEMU;
- Many sub-projects:
  - Eclipse integration,
  - creation of a SDK for your distribution (ADK[1]),
  - EGlibc...

[1] ADK: ***A**pplication **D**evelopment **K**it*

Optimisation and debug

Remote debugging with GDB[1]

- Symbolic debugging;
- Target process remotely controlled from host with GDB (or graphical overlay like DDD[2]);
- Two options on the target side:
  - `gdbstub`: collection of hooks and handlers available in the target firmware or kernel allowing to debug it remotely by manipulating the hardware, or
  - `gdbserver`: small application installed on the target and allowing to remotely debug an application using OS services (`ptrace()` Unix system call).

[1]GDB: ***G**NU **De**Bugger*
[2]DDD: ***D**ata **D**isplay **D**ebugger*

gdbserver

- It allows to debug applications only but it is simpler and more common in the Linux world;
- The `gdbserver` part, available on the target, retrieves debugging commands from the GDB on host and send the results back;
- Several communication links are available (serial, TCP/IP...);
- Example of connection via TCP/IP:

target> `gdbserver :2222 hello`

host> `arm-linux-gdb hello` or `ddd -gdb -debugger arm-linux-gdb hello`

(gdb) `target remote 192.168.0.10:2222`

(gdb) `list`

(gdb) `break 10`

(gdb) `cont`

strace

- Tool to intercept (uses `ptrace()`) all system calls made by a process and display it in a human-readable manner;
- Ability to filter intercepted system calls (eg `strace -e trace=open,close,read ls`);
- Can be installed on the target during developments and removed from the final version;
- Located on the border between user space and kernel space, it helps to figure out which of application or kernel (rare;-) misbehaves.

LTTng[1] / *SystemTap*

- Full-featured analysis softwares for system and kernel;
- Includes a core part (traces capture) and a user part (traces acquisition);
- Traces from the target can be analyzed on the host with dedicated graphics applications;
- Allows comprehensive analysis of timing issues, inter-process communication, user/kernel timing...

[1] LTT: ***L**inux **T**race **T**oolkit **n**ext **g**eneration*

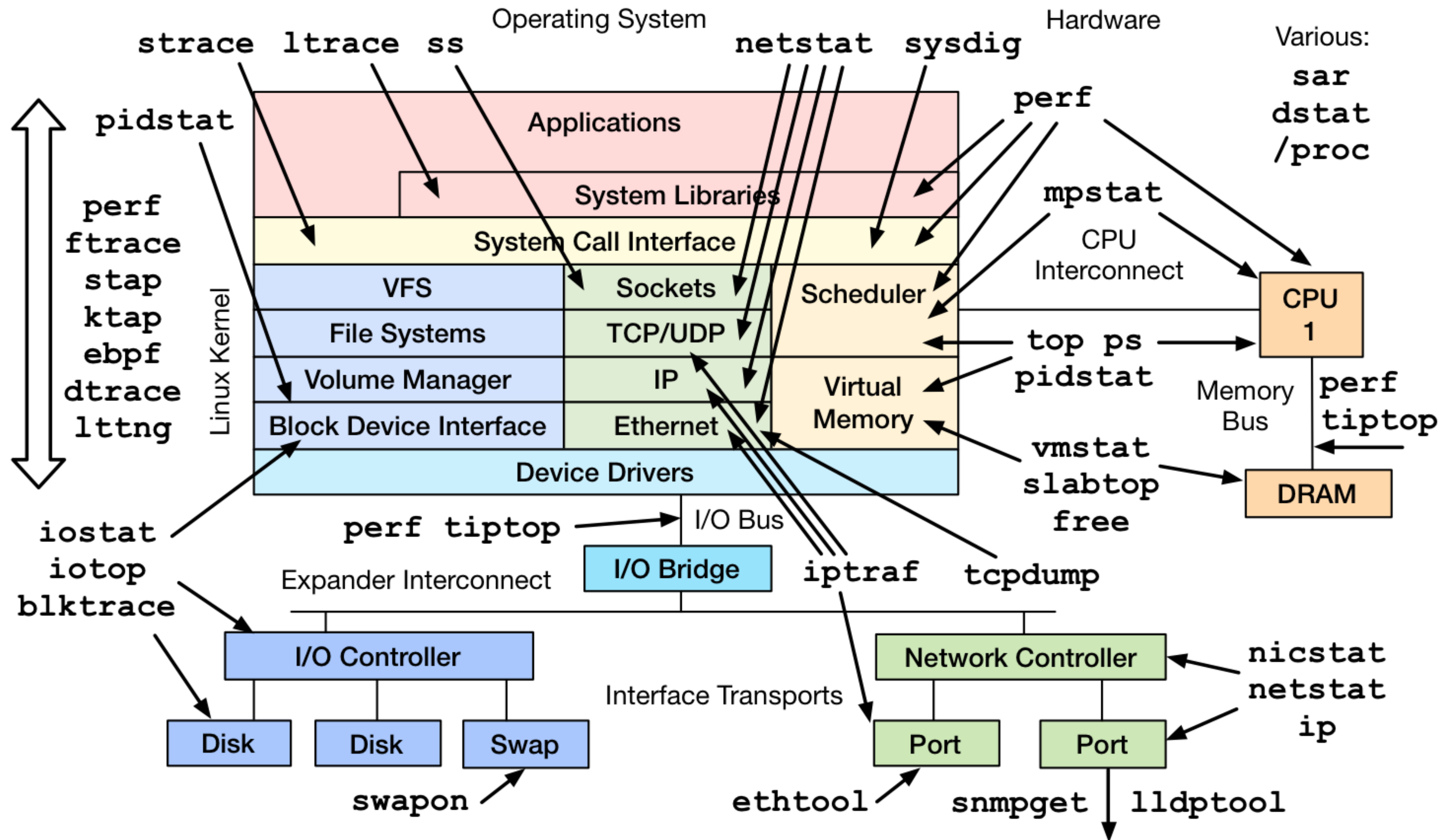
Profiling

- Technique of making statistics on execution times of different parts of an application for debugging and/or optimization;
- The Linux kernel has its own profiling system (boot option `profile=n`, `/proc/profile` and tool `readprofile`) that looks at the instruction pointer at each system timer interrupt and then updates statistics on most used kernel functions;
- For applications, GCC has also its own profiling system (`-pg` option) that makes a statistic-file during application runtime (exploitable later with the `gprof` tool).

Hardware debug

- ROM emulator ⇒ RAM component (*overlay RAM*) drived with a serial link it allows quick code upload on the target and setting breakpoints on it;
- ICE[1] ⇒ physical CPU emulator that take the place of the CPU on the board and can simulate up to I/O;
- JTAG[2] ⇒ can be used as a OCD[3] that allows to set breakpoints and to read/write registers on microcontrollers but often limited to on-chip memory programming;

[1] ICE: ***I****n* ***C****ircuit* ***E****mulator*
[2] JTAG: ***J****oint* ***T****est* ***A****ction* ***G****roup*
[3] OCD: ***O****n* ***C****hip* ***D****ebugger*



Linux Performance Observability Tools
Operating System
Hardware
Various:
sar
dstat
/proc
strace ltrace ss
netstat sysdig
perf
pidstat
perf
ftrace
stap
ktap
ebpf
dtrace
lttng
Linux Kernel
Applications
System Libraries
System Call Interface
VFS
Sockets
Scheduler
File Systems
TCP/UDP
Volume Manager
IP
Virtual Memory
Block Device Interface
Ethernet
Device Drivers
mpstat
CPU Interconnect
CPU 1
top ps
pidstat
Memory Bus
perf
tiptop
vmstat
slabtop
free
DRAM
iostat
iotop
blktrace
perf tiptop
I/O Bus
I/O Bridge
Expander Interconnect
iptraf
tcpdump
I/O Controller
Network Controller
nicstat
netstat
ip
Interface Transports
Disk
Disk
Swap
Port
Port
swapon
ethtool
snmpget
lldptool
Brendan Gregg 2014

Software emulation and virtualization

- QEMU (http://www.bellard.org/qemu/) ⇒ multi-platform CPU emulator (x86, ARM, SPARC, PowerPC) with two running modes (full system or Linux application);
- ARMulator (http://www.gnu.org/software/gdb/) ⇒ GNU debugger (GDB) extension that can emulate many different ARM cores (*big endian*, *little endian* and *thumb*);
- Xcopilot (http://www.uclinux.org/pub/uClinux/utilities/) ⇒ full PalmPilot emulator (68k, timers, serial port, touchscreen...), it was used for engineering the first μClinux version;
- POSE[1] (http://sf.net/projects/pose/) ⇒ multi-platform Palm PDA emulator, it is a rework of the Palm Copilot;

[1]POSE: ***Palm OS Emulator***

- UML[1] (http://user-mode-linux.sf.net/) ⇒ Linux kernel running over another Linux kernel, it allows to run multiple kernels as standard processes of a host kernel;
- VMware (http://www.vmware.com/) / VirtualBox (http://www.virtualbox.org/) ⇒ multi-platform virtual machines (commercial/free) that can emulate a full x86/PC (CPU, BIOS, drives, network...) and that can run many of the OS for this arch;
- Bochs (http://bochs.sf.net/) ⇒ free multi-platform x86/PC emulator (LGPL license);
- MAME[2] (http://www.mame.net/) ⇒ arcade emulator that emulates many old processors (z80, M6809, 68k...) that are sometimes still in use in the embedded world.

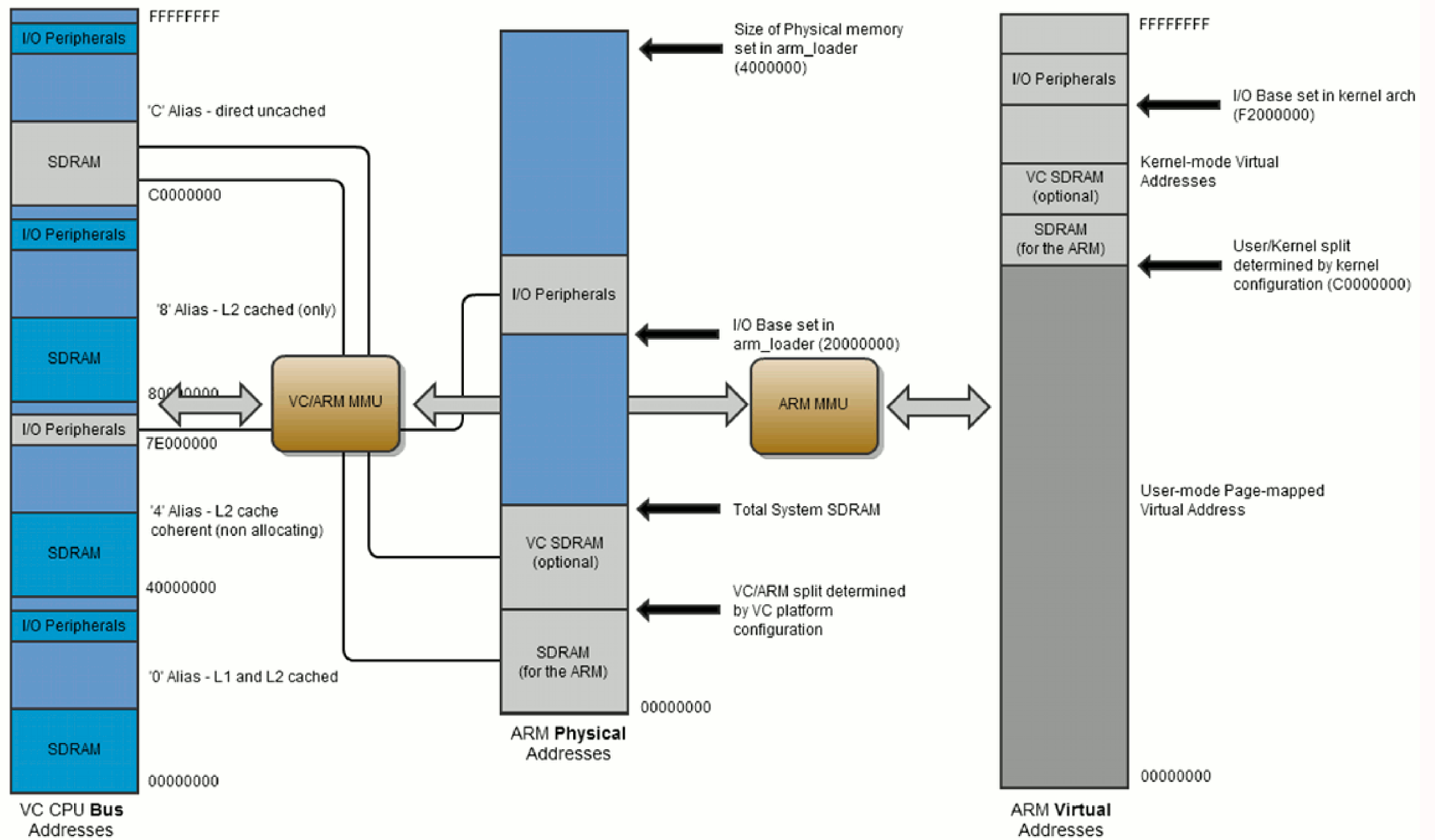
[1] UML: ***U****ser* ***M****ode* ***L****inux*

[2] MAME: ***M****ultiple* ***A****rcade* ***M****achine* ***E****mulator*

Case study

Raspberry Pi

- Implementation of an embedded system on the *Raspberry Pi* SOB (ARM11, SD Flash, 256MB SDRAM and 100 Mb Ethernet controller):
  - setup of the Raspbian distribution,
  - cross-compilation,
  - remote debugging with `gdbserver`,
  - web server and PHP,
  - system from scratch with *buildroot*.

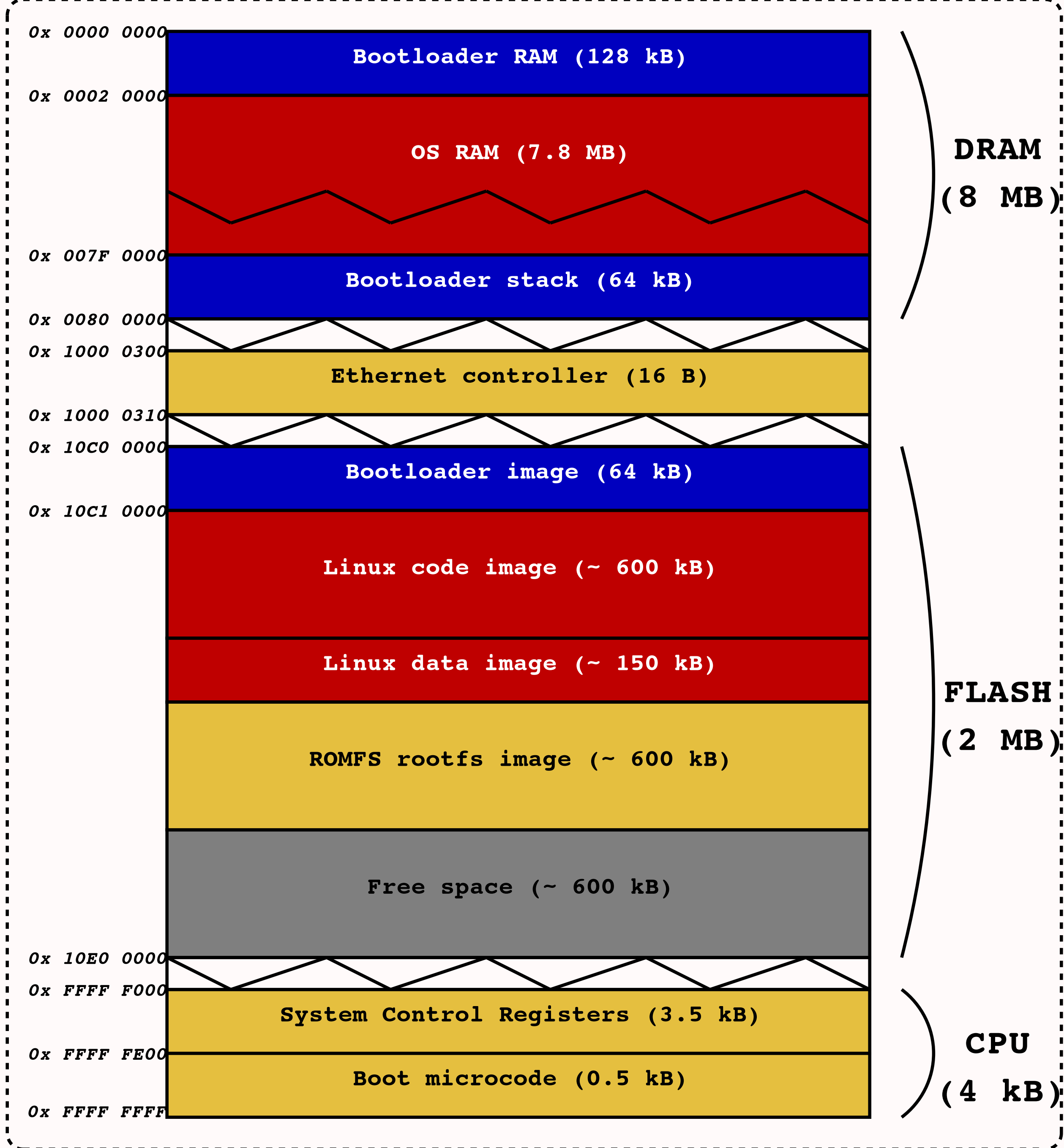


I/O Peripherals
SDRAM
I/O Peripherals
SDRAM
I/O Peripherals
SDRAM
I/O Peripherals
SDRAM
FFFFFFFF
'C' Alias - direct uncached
C0000000
'8' Alias - L2 cached (only)
80000000
7E000000
'4' Alias - L2 cache coherent (non allocating)
40000000
'0' Alias - L1 and L2 cached
00000000
VC CPU Bus Addresses
VC/ARM MMU
I/O Peripherals
VC SDRAM (optional)
SDRAM (for the ARM)
ARM Physical Addresses
Size of Physical memory set in arm_loader (4000000)
I/O Base set in arm_loader (20000000)
Total System SDRAM
VC/ARM split determined by VC platform configuration
00000000
ARM MMU
I/O Peripherals
VC SDRAM (optional)
SDRAM (for the ARM)
FFFFFFFF
I/O Base set in kernel arch (F2000000)
Kernel-mode Virtual Addresses
User/Kernel split determined by kernel configuration (C0000000)
User-mode Page-mapped Virtual Address
00000000
ARM Virtual Addresses

μCsimm

- Implementation of an embedded system on the *μCsimm* SOB (DragonBall EZ, 2 MB Flash, 8 MB DRAM and 10 Mb Ethernet controller):
  - setup of the μClinux distribution,
  - cross-compilation,
  - remote debugging with `gdbserver`,
  - web server and CGI[1].

[1] CGI: ***C**ommon, **G**ateway **I**nterface*


uCsimm physical memory map
0x 0000 0000
Bootloader RAM (128 kB)
0x 0002 0000
OS RAM (7.8 MB)
0x 007F 0000
Bootloader stack (64 kB)
0x 0080 0000
0x 1000 0300
Ethernet controller (16 B)
0x 1000 0310
0x 10C0 0000
Bootloader image (64 kB)
0x 10C1 0000
Linux code image (~ 600 kB)
Linux data image (~ 150 kB)
ROMFS rootfs image (~ 600 kB)
Free space (~ 600 kB)
0x 10E0 0000
0x FFFF F000
System Control Registers (3.5 kB)
0x FFFF FE00
Boot microcode (0.5 kB)
0x FFFF FFFF
DRAM
(8 MB)
FLASH
(2 MB)
CPU
(4 kB)

References

Books

- **Building Embedded Linux Systems** - Karim Yaghmour (http://www.embeddedtux.org/);
- ***Linux Embarqué*** - Pierre Ficheux (http://pficheux.free.fr/);
- **Embedded Linux** - John Lombardo;
- **Embedded Linux** - Craig Hollabaugh;
- **Linux for Embedded and Real-time Applications** - Doug Abbott.

Portals

- LinuxGizmos (http://linuxgizmos.com/);
- Embedded Linux Wiki (http://elinux.org/);
- The Linux Foundation (http://www.linuxfoundation.org/);
- The Linux Documentation Project (http://www.tldp.org/).

Web

- French courses from Patrice Kadionik at ENSEIRB (http://www.enseirb.fr/ kadionik/);
- Bill Gatliff homepage (http://billgatliff.com/);
- Nicolas Ferre homepage (http://nferre.free.fr/);
- The μClinux directory (http://uclinux.home.at/);
- Embedded Debian (http://www.emdebian.org/);
- Filesystems (http://en.wikipedia.org/wiki/List_of_file_systems).

Hardware

- OpenHardware (http://www.openhardware.net/);
- LART (http://www.lart.tudelft.nl/);
- OpenCores (http://www.opencores.org/);
- GumStix (http://www.gumstix.com/);
- ArmadeouS (http://www.armadeus.com/);
- Raspberry Pi (http://www.raspberrypi.org/).

The end