

Séance 6

BeagleBone Black et Linux embarqué



Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

Objectifs

- Linux embarqué et ses spécificités

Notions de système informatique et d'exploitation

- Programmation d'un système embarqué

Installation et démarrage d'un système et support hardware

- Le système embarqué BeagleBone Black (BBB)

- Caractéristiques hardware et software du système embarqué
- Électronique numérique avec la BBB et Linux embarqué
- Développement de drivers

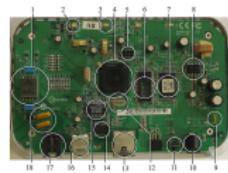


Linux embarqué

Type de système informatique

■ Plusieurs types de systèmes informatiques

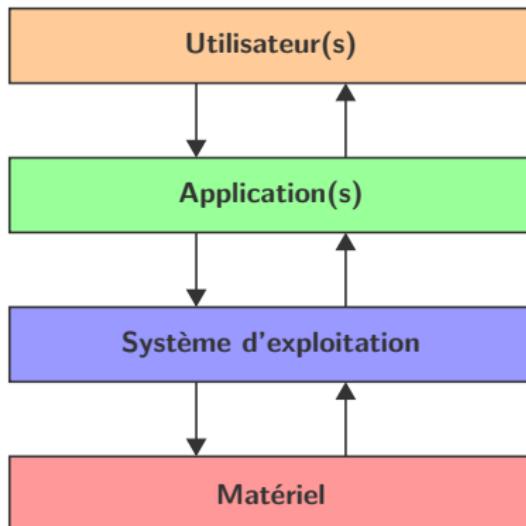
- PC (personal computer)
- Mainframe
- Station de travail (workstation)
- Système mobile (portable, tablette, smartphone...)
- Système embarqué



Système informatique (1)

- Système informatique peut être vu comme quatre couches

Utilisation de services inférieurs pour en offrir des supérieurs



Système informatique (2)

- Le **matériel** fournit les ressources de calcul

Processeur (CPU), mémoire, périphériques d'entrée-sortie (E/S)

- Le **système d'exploitation** coordonne l'utilisation du matériel

Coordination entre les applications et les utilisateurs

- Les **applications** résolvent les problèmes des utilisateurs

Traitement de texte, compilateur, jeu vidéo, navigateur web...

- Les **utilisateurs** utilisent le système informatique

Personne, machine, autre ordinateur



Système d'exploitation

- **Couche logicielle** intermédiaire entre l'utilisateur et le hardware
Fournit aux programmes utilisateurs une interface avec le matériel
- **Abstractions** des installations matérielles sous-jacentes
Processus, fichier, socket, RPC, RMI, librairie de dessin, fenêtre
- Différents **buts**
 - **Pratique** : utilisation aisée par les utilisateurs (PC, mobile)
 - **Efficace** : optimiser l'utilisation du hardware (mainframes)
 - **Évolution** : ajout de nouvelles fonctions systèmes

Linux (1)

- Linux paru pour la première fois en été 1991

Hobby de l'informaticien Finlandais Linus Torvalds

- Développement de distributions Linux par la communauté

Installation possible sans devoir être un expert technique

- Linux se retrouve dans tout et n'importe quoi

Gadget de poche, superordinateur, DTV receiver, smartphone...



redhat.



Free Software Foundation (FSF)

- FSF fondée par **Richard Stallman** le 4 octobre 1985

Support du mouvement du logiciel libre

- Écriture de logiciels libres pour le **projet GNU**

Notamment GNU GCC Compiler, un compilateur C



- Existait une bonne décennie **avant l'arrivée** de Linux

Nécessité des outils pour compiler et utiliser Linux



Linux (2)

- Kernel d'un système d'exploitation écrit par Linus Torvalds 
Propose des facilités de base pour construire un système dessus
- Démarré par un bootloader ou par un firmware système
Jamais éteint (même quand le système est low-powered mode)
- Mauvaise utilisation du terme Linux aujourd'hui
Le kernel Linux, le système Linux, une distribution Linux



<https://github.com/torvalds/linux> (version 4.15 au 28 janvier 2018)

Distribution Linux

- Une **distribution Linux** est un système d'exploitation
 - Un ensemble de logiciels
 - Basé sur le kernel Linux
 - Souvent assorti d'un gestionnaire de paquets
 - Souvent accompagné d'un gestionnaire de fenêtres
- Trois **grandes familles** de distribution Linux
 - Red Hat Enterprise Linux (RHEL)
 - SuSE Linux Enterprise Server (SLES)
 - Debian GNU/Linux

Linux embarqué



- Linux embarqué est une distribution pour système embarqué

Pas de version embarquée du kernel Linux

- Système ou distribution Linux embarquée

Ajout d'outil de développement, compilateur, debugger...



Linux temps-réel

- Projet **RTLinux** lancé en 1996 par Michael Barabanov
Fournir des temps de réponse déterministes sous Linux
- Vendu à **Wind River** au début 2007 (racheté par Intel en 2009)
Société spécialisée en OS pour système embarqué
- OS temps réel **VxWorks** notamment utilisé par la NASA
Industrie aéronautique, automobile, transport, télécommunication

Types de Linux embarqué (1)

- Classification sur base de **quatre critères**

Linux ne tourne pas sur une architecture en-dessous de 32 bits

- **Taille** du système embarqué



- Taille physique détermine les capacités matérielles
- Vitesse CPU, taille de RAM, taille du stockage permanent
- Trois catégories

Taille	CPU	ROM	RAM
Petit	Low-powered	≥ 4 Mio	8–16 Mio
Moyen	Medium-powered	≥ 32 Mio	64–128 Mio
Grand	Powerful ou plusieurs		Large quantité

Types de Linux embarqué (2)

■ Contrainte de temps du système embarqué

- Différence entre soft et hard real-time
- Deux types de contraintes

Type	Description
Strict	Le système doit répondre endéans une fenêtre temporelle
Souple	Le temps de réponse n'est pas critique

■ Mise en réseau du système embarqué



Possibilité ou non pour le système d'être connecté à un réseau

■ Interaction entre le système embarqué et l'utilisateur

Depuis un écran LCD à des LEDs ou aucune interface du tout

Pourquoi Linux ? (1)

- Qualité et fiabilité du code

Du kernel et des logiciels fournis avec la distribution

- Qualité : modulaire, structuré, lisible, extensible, configurable
- Fiabilité : prévisible, récupération après erreur, longévité

- Disponibilité du code

Accès sans restriction au code de Linux

- Large support hardware

Drivers disponibles pour de très nombreux dispositifs hardware

Pourquoi Linux ? (2)

- Protocoles de **communication** et softwares standards

Par exemple, support Samba pour être dans réseau Windows

- **Outils** disponibles

De nombreux outils existent déjà, pas besoin de réinventer la roue

- Large **support** de la communauté

- **Licence**, indépendance des vendeurs, cout

On peut utiliser, modifier et redistribuer suivant les mêmes droits

Développement



Hôte et connexion avec la cible (1)



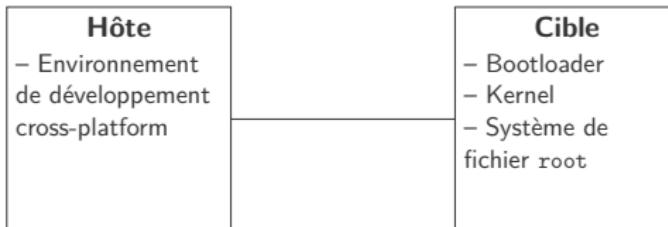
- Station de travail Linux comme machine hôte

$\geq 1\text{--}2 \text{ Gio RAM}$, $\geq 2 \text{ GHz CPU}$, un maximum d'espace disque



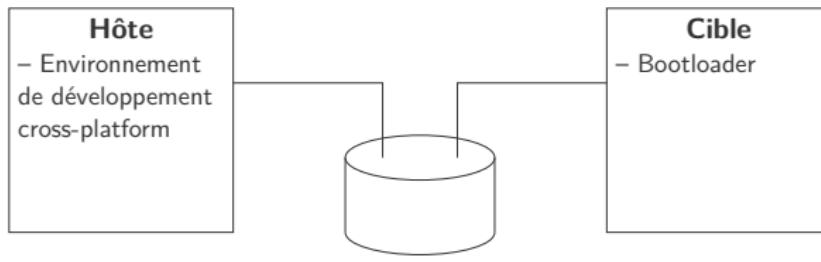
- Connexion permanente avec un câble physique

- Typiquement avec un lien Ethernet ou série (RS232)
- Possibilité de monter les systèmes de fichier de la cible
- Permet de faire du debug en live



Hôte et connexion avec la cible (2)

- Stockage amovible écrit par l'hôte et transféré vers la cible
 - CompactFlash, carte MMC... chargé par bootloader minimal
 - Possibilité d'être inséré et facilement retiré sur la cible
 - Très pratique dans les phases initiales du développement



- Bootloader secondaire
- Kernel
- Système de fichier root

Hôte et connexion avec la cible (3)



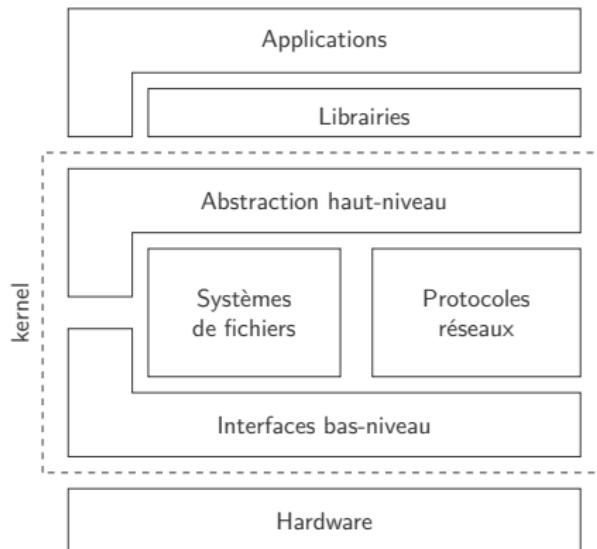
- **Développement autonome** sur le système embarqué
 - Le système cible contient tout ce qu'il faut pour développer
 - Station de travail qui tourne sur un système embarqué
 - Distribution classique peut tourner sur ces systèmes

Cible
– Bootloader
– Kernel
– Système de fichier root
– Environnement de développement natif

Architecture

■ **Architecture** d'un système Linux générique

Système logiciel architecturé en couches et avec un kernel



Hardware et kernel

- Caractéristiques hardware nécessaires pour un système Linux
 - Au moins un CPU 32 bits équipé d'un MMU
 - Suffisamment de RAM pour héberger le système
 - Capacités E/S minimales (notamment pour le debugging)
 - Possible de charger système de fichiers `root` (disque, réseau...)
- Le kernel fait l'interface avec le hardware
 - Offre des abstractions de haut niveau à la couche user
 - Interfaces de bas niveau spécifique au hardware (même API)

Démarrage



- Trois composants software impliqués au **démarrage** de Linux
 - **Bootloader** : initialisation bas-niveau hardware, saut au kernel
 - **Kernel** : code de démarrage, puis générique `start_kernel()`
 - **Processus init** : processeur initial exécuté en mode user
- Le **bootloader** dépend largement du hardware cible

Tout comme le code de démarrage du kernel
- `start_kernel()` **initialise** fonctions de haut niveau du kernel

Et monte système de fichiers root, puis démarre init



init/main.c

```
493 asmlinkage __visible void __init start_kernel(void)
494 {
495     char *command_line;
496     char *after_dashes;
497
498     set_task_stack_end_magic(&init_task);
499     smp_setup_processor_id();
500     debug_objects_early_init();
501
502     /*
503      * Set up the the initial canary ASAP:
504      */
505     boot_init_stack_canary();
506
507     cgroup_init_early();
508
509     local_irq_disable();
510     early_boot_irqs_disabled = true;
511
512     /*
513      * Interrupts are still disabled. Do necessary setups, then
514      * enable them
515      */
516     boot_cpu_init();
517     page_address_init();
518     pr_notice("%s", linux_banner);
519     setup_arch(&command_line);
520     mm_init_cpumask(&init_mm);
521     setup_command_line(command_line);
522     setup_nr_cpu_ids();
523     setup_per_cpu_areas();
524     smp_prepare_boot_cpu(); /* arch-specific boot-cpu hooks */
525
526     build_all_zonelists(NULL, NULL);
527     page_alloc_init();
528
529     pr_notice("Kernel command line: %s\n", boot_command_line);
530     parse_early_param();
531     after_dashes = parse_args("Booting kernel",
532                             static_command_line, __start__param,
533                             __stop__param - __start__param,
534                             -1, -1, NULL, &unknown_bootoption);
535
536     if (!IS_ERR_OR_NULL(after_dashes))
537         parse_args("Setting init args", after_dashes, NULL, -1, -1,
```

Configuration de boot

- Bien choisir le **type de configuration** de boot

Influence le choix de bootloader, sa configuration et l'hôte

- Exemples de **contraintes**

- Hôte fournit accès à la cible pour configuration par réseau
- Tous les bootloaders savent pas charger kernel depuis un disque

- **Trois types** de setups pour bootstraper un système Linux

Média de stockage solid-state, disque, réseau

Média de stockage *solid-state*

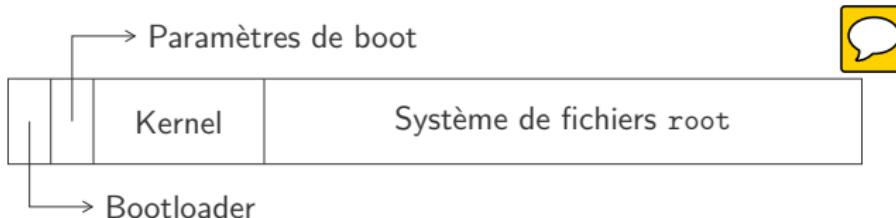


- Bootloader initial stocké sur un **stockage solid-state**

Avec configuration, kernel et système de fichiers root

- Mémoire typiquement découpée en **quatre parties**

- Pas une obligation (parfois kernel dans système fichiers root)
- Mécanisme de protection spécial pour la zone du bootloader



Disque



- Système de fichiers root et kernel sur un **disque**

Configuration classique sur les stations de travail et serveurs

- **Bootloader initial** charge un second bootloader ou le kernel

Souvent limité à un secteur de boot de 512 octets

- Permet de **tester** plusieurs kernels et configurations

Un des systèmes de fichier du disque sera utilisé comme root

Réseau



- Chargement système de fichiers et/ou kernel **depuis le réseau**
 - La cible doit avoir un lien réseau disponible
 - Le kernel peut être sur média de stockage *solid-state* ou disque
- Utilisation de **protocoles** de chargement
 - *Trivial File Transfer Protocol* (TFTP) charge le kernel
 - *Network File System* (NFS) monte système de fichiers `root`
- Facilite le **partage de données** entre développeurs

Aussi, les mises à jour sont faites à distance par un seul point

Organisation de la mémoire

- **Espace d'adresses** physique et virtuel (adresses logiques)

Espace dans la mémoire physique ou dans le kernel

- **Périphériques hardwares** accessibles dans l'espace physique

Accès restreint voir invisible dans l'espace logique

- Utilité du **plan d'organisation** de la mémoire

- **Physique** : comment configurer kernel et développer drivers
- **Virtuel** : comprendre et débugguer les applications

Support hardware

- Linux embarqué supporte plusieurs **devices hardware**
Architecture de processeurs et autres composants hardwares
- **Deux types** de drivers pour les périphériques
 - Driver propriétaire
 - Driver open-source

Architecture des processeurs (1)

- Architectures supportées se retrouvent dans le **dossier arch**

Plus de 20 architectures différentes supportées

- **Huit** typiquement utilisés pour Linux embarqué

ARM, AVR32, Intel x86, M32R

MIPS, Motorola 68000, PowerPC et Super-H

- **uClinux** pour des systèmes sans MMU

Notamment chez Analog Devices ou Xilinx pour des FPGA

arch

This repository Search

Pull requests Issues Gist

Watch 4,550 Star 30,503 Fork 12,025

torvalds / linux

Code Pull requests 96 Pulse Graphs

Branch: master linux / arch /

New file Upload files Find file History

torvalds Merge branch 'sched-core-for-linus' of git://git.kernel.org/pub/scm/l... ... Latest commit d4e7961 12 hours ago

...

⚠ Failed to load latest commit information.

alpha	dma-mapping: always provide the dma_map_ops based implementation	2 months ago
arc	arc: SMP: CONFIG_ARC_IPI_DBG cleanup	20 days ago
arm	Merge branch 'sched-core-for-linus' of git://git.kernel.org/pub/scm/l... ...	12 hours ago
arm64	Merge branch 'mm-readonly-for-linus' of git://git.kernel.org/pub/scm/l... ...	14 hours ago
avr32	avr32: Set IORESOURCE_SYSTEM_RAM flag for System RAM	2 months ago
blackfin	dma-mapping: always provide the dma_map_ops based implementation	2 months ago
c6x	locking/lockdep: Eliminate lockdep_init()	a month ago
cris	Merge branch 'akpm' (patches from Andrew)	2 months ago
frv	dma-mapping: always provide the dma_map_ops based implementation	2 months ago
h8300	Merge branch 'akpm' (patches from Andrew)	2 months ago
hexagon	dma-mapping: always provide the dma_map_ops based implementation	2 months ago
ia64	ia64: Set System RAM type and descriptor	2 months ago
m32r	Merge tag 'v4.5-rc6' into core/resources, to resolve conflict	11 days ago
m68k	m68k/defconfig: Update defconfigs for v4.5-rc1	a month ago
metag	dma-mapping: always provide the dma_map_ops based implementation	2 months ago

Architecture des processeurs (2)

- Advanced RISC Machine (ARM)

- Ne construisent pas les processeurs, mais vendent les designs
- Même ensemble d'instructions pour tous les ARM
- Langage d'assemblage et binaire identiques par révision

- AVR32



Architecture 32 bits designé par Atmel Corp. (2006)

- Intel x86

- Démarre avec le 386 lancé par Intel en 1985
- Deux grands distributeurs de x86 : Intel et AMD
- Petite fraction seulement du marché des systèmes embarqués

Architecture des processeurs (3)

■ M32R

- Architecture 32 bits par Renesas Tech. (2003)
- Tourne sur un système basé sur des FPGAs

■ MIPS

- Fils spirituel de John Hennessey
- Silicon Graphics, Inc., Nintendo 64, Sony Playstation 1 et 2
- Plusieurs jeux d'instructions selon les modèles

■ Motorola 68000

- Connu sous le petit nom m68k
- Oublié de nos jours au profit de ARM, MIPS, SH et PowerPC

Architecture des processeurs (4)

■ PowerPC (PPC)

- Collaboration entre Apple, IBM et Motorola (alliance AIM)
- Utilisé sur des Mac de Apple et des stations de travail IBM
- Architecture la mieux supportée avec i386 et ARM pour Linux
- Yellow Dog Linux est une distribution uniquement pour PPC

■ SuperH (SH)

- Inventé par Hitachi au début des années 1990
- Données 32 bits en interne et longueurs de bus externe variées
- Pas de MMU dans les premières versions

Bus et interfaces

- **Structure** qui connecte le CPU aux périphériques du système
Support par Linux varié selon le bus ou l'interface
- Plusieurs **bus et interfaces** supportés
 - PCI/PCI-X/PCIe, ExpressCard
 - PC/104, PC/104-Plus/PCI-104, PCI/104-Express
 - CompactPCI/PCIe, SCSI/iSCSI, USB, IEEE1394 (FireWire)
 - InfiniBand, GPIB 
 - I²C, I/O, Serial Port, Parallel Port, Modem, Data Acquisition
 - Clavier, Souris, Écran, Son, Imprimante

Stockage

- Stockage permanent notamment pour le boot process
Même stockage utilisé ensuite pour les opérations de l'OS
- Memory Technology Devices (MTD)
Par exemple ROM ou mémoire flash NOR/NAND
- Standards d'interface avec le stockage
PATA, SATA, ATAPI (IDE)
- Devices qui ne sont pas MTD
CompactFlash, SD, stick USB

Réseau General-Purpose

- Système embarqué attachés à des réseaux general-purpose
Même support qu'avec Linux pour les stations de travail
- Plusieurs devices hardware pour établir une connexion
 - Ethernet de 10–100 Mbps à 10 Gbps
 - Infrared Data Association (IrDA) à 4 Mbps sur 1 mètre
 - IEEE 802.11A/B/G/N jusque 248 Mbps
 - Bluetooth

Réseau Industrial-Grade

- Contrôle industriel et automatisation se basent sur des réseaux
 - Ethernet et les autres ne sont pas adaptés à l'environnement
 - Ethernet standard vulnérable aux EMI et RFI
- Avantages des *fieldbuses*

Moins de câbles, plus de modularité, auto-configuration...
- Deux principaux réseaux industriels supportés par Linux
 - Controller Area Network (CAN) pour l'automobile au départ
 - Modbus pour données de contrôle vers senseurs avec RS232

Monitoring du système

- Des **défaillances** sont parfois inévitables

Pouvoir les détecter et s'en remettre grâce au monitoring

- Deux types de **monitoring** proposé par Linux

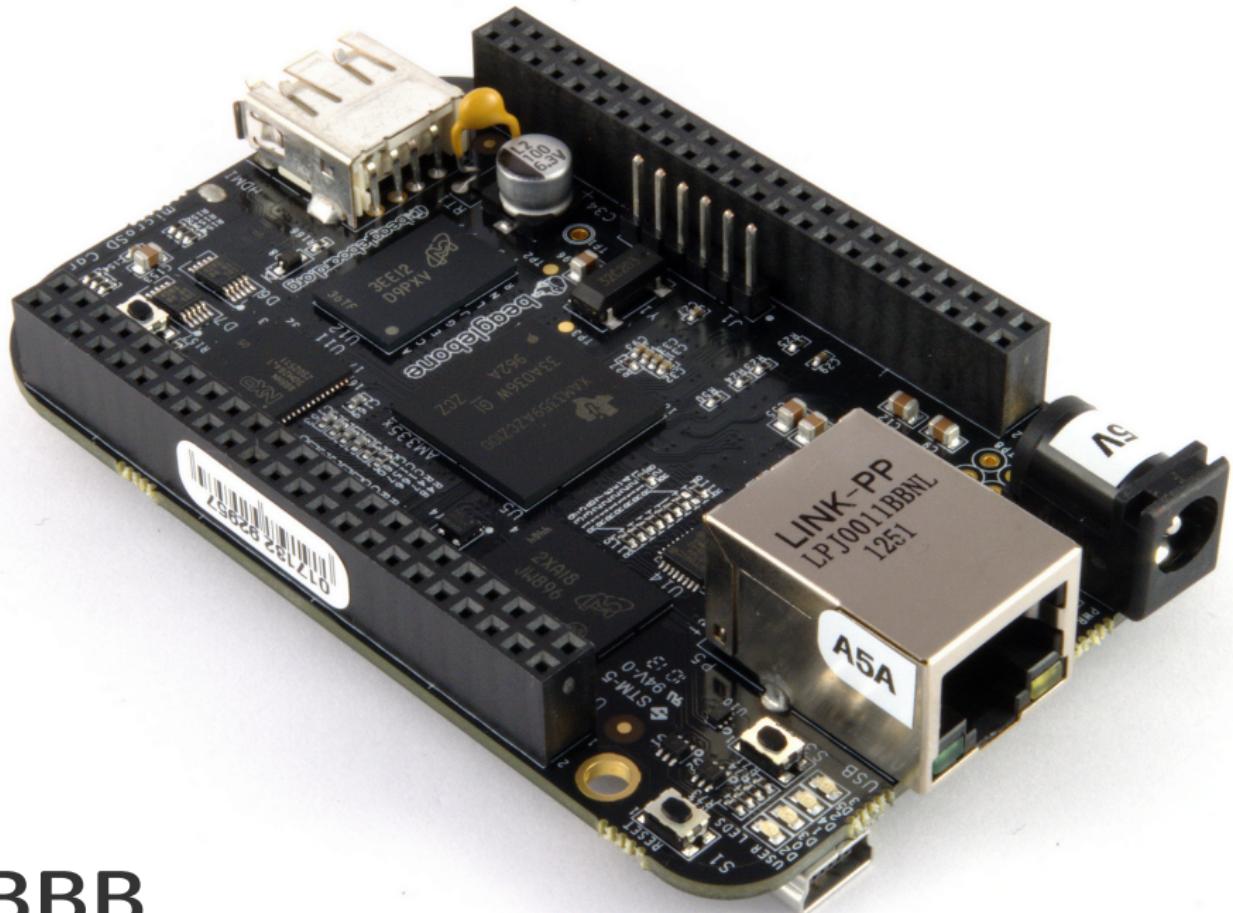
- **Watchdog timers** (hardware/software)

Réinitialisation périodique ou reboot



- **Hardware health monitoring** (hardware spécifique)

Fournit de l'information sur l'état de santé physique du système



BBB

Caractéristiques hardware



- Trois **unités de calcul**
 - Processeur ARM 1 GHz Cortex-A8, 512 Mio de RAM DDR3
 - Co-Processeur ARM Cortex-M3 pour gérer l'énergie
 - PRUSS
- Plusieurs **entrée/sortie** pour communiquer avec l'extérieur
Ethernet, USB, extensions, Mini USB, MicroSD, Micro HDMI
- Deux **niveaux de tension** pour les entrées/sorties
Digital I/O à 3.3 V et analog I/O à 1.8 V



Spécificités BeagleBone Black

- Entrée/sortie complémentaires

Port Micro HDMI, header série avec support FTDI TTL-232

- Mémoire **flash onboard** de type eMMC



Mémoire flash + contrôleur (embedded Multi-Media Controller)

- Présence d'un **boot switch** pour choisir mode démarrage

Longue pression pour depuis Micro SD au lieu de eMMC

Documentation

- Site web de la BBB avec un guide “*Getting Started*”
http://www.beagleboard.org
- Le *System Reference Manual* (SRM) officiel
Description de tout le hardware en 125 pages
- Le *Technical Reference Manual* (TRM) du TI ARM335x
Tout ce que vous avez toujours voulu savoir en 4593 pages !

Pour qui ?

- Transformer un concept en **produit électronique interactif**

Intégration software haut niveau avec électronique bas niveau

- **Grande communauté** avec variété d'outils à disposition

Environnements de développement, Bonescript, outils...

- **Personnalisation** de la plateforme BeagleBone

Tant au niveau hardware que software

Pourquoi ?

- Réseau built-in et **accès à distance** à la carte
Port Ethernet et outils de communication (FTP, SSH, NTP)
- Organisation des fichiers et accès uniforme aux périphériques
Système de fichiers, et tout est fichier sous Unix/Linux 
- Plusieurs langages de programmation et **multitasking**
Partage du processeur entre plusieurs processus

Pour quoi ?

- **Intégration** software haut niveau avec électronique bas niveau
Automated home management system, robot, affichage intelligent, réseau de senseurs, vending machine, etc.
- Coopération et utilisation des **outils disponibles sur Linux**
Serveur web Apache pour intégration avec le web, par exemple
- Exploitation de **périphériques USB** à l'aide de drivers
Caméra, adaptateurs WiFi, etc.

Pas pour quoi ?

- Pas possible de gérer contrainte **temps réel hard**



Linux n'est pas capable d'assurer une telle contrainte

- Pas possible de faire du **traitement vidéo** haute définition

Pas fait pour tourner XBMC, par exemple



Distribution Linux embarqué

- Deux **distributions spécifiques** par défaut avec la BBB

Linux embarqués spécialement conçu pour la BBB

- Distribution **Angström** Linux

Construit avec OpenEmbedded, différent des classiques

- Distribution **Debian** Linux

Plus d'activités liées au hardware

Connexion USB

- Connexion possible à la BBB par USB (*USB host*)
Permet d'accéder à deux fonctions différentes
- Stockage de masse sur les 4 Go de mémoire flash (eMMC)
Drivers spécifiques pour supporter l'USB Mass Storage (UMS)
- Interface de réseau virtuelle de type Ethernet
Remote Network Driver Interface Specification (RNDIS)

Connexion Serial over USB

- Utilisation **header série** sur BeagleBone Black

Connexion en mode texte uniquement, à l'aide d'un terminal

- Utilisation d'un câble ou adaptateur **FTDI TTL-232**

- Compagnie écossaise privée spécialisée en technologie USB
- Conversion de transmission série RS-232 ou TTL en USB

Exécution de commandes

- Exécution de **commandes** sur la BBB après connexion
 - Sur un routeur via Ethernet (SSH)
 - À un ordinateur via USB (disque monté, SSH ou série)
- **Protocole SSH** (Secure Shell)

Connexion sécurisée à un shell sur une machine distante

Système de fichiers

■ Principaux dossiers du *système de fichiers*

Dossier	Description
/bin	Programmes et commandes pour l'utilisateur
/boot	Fichiers nécessaires pour le démarrage
/dev	Fichiers qui représentent des périphériques sur votre système
/etc	Fichiers de configuration
/home	Dossiers des utilisateurs
/lib	Librairies systèmes et drivers
/media	Localisation de médias amovibles (Stick USB, carte SD...)
/proc	Fichiers qui représentent des informations sur le système
/sbin	Programmes pour la maintenance du système
/sys	Fichier pour accéder au hardware de la BBB
/tmp	Fichiers temporaires
/usr	Programmes disponibles pour tous les utilisateurs
/var	Fichiers logs

Mise à jour

- Mettre à jour les différents package d'un système

Rechercher les nouvelles versions et les installer

- Deux façons de procéder selon la distribution
 - Téléchargement et copie de binaires précompilés
 - Téléchargement et recompilation des sources

```
$ opkg update  
$ opkg upgrade
```

Électronique numérique (1)

- La BBB possède **deux rangées** de pins (P8 et P9)

Il y a 46 pins par rangées (dont 1, 2, 45 et 46 sont numérotées)

- Plusieurs **fonctions différentes** pour les pins

- Contrôler un écran LCD
- Lire un senseur
- Communiquer avec d'autres dispositifs électroniques
- ...

- Possibilité de **changer le mode** de certaines pins



Électronique numérique (2)

■ Possibilité de **changer le mode** de certaines pins

Deux fois 46 pins, avec huit modes de configuration (SRM, p.84)

PIN	PROC	NAME	MODE0	MODE1	MODE2	MODE3 GND	MODE4	MODE5	MODE6	MODE7
1,2										
3	R9	GPIO1_6	gpmc_ad6	mmc1_dat6						gpio1[6]
4	T9	GPIO1_7	gpmc_ad7	mmc1_dat7						gpio1[7]
5	R8	GPIO1_2	gpmc_ad2	mmc1_dat2						gpio1[2]
6	T8	GPIO1_3	gpmc_ad3	mmc1_dat3						gpio1[3]
7	R7	TIMER4	gpmc_adrv_ale		timer4					gpio2[2]
8	T7	TIMER7	gpmc_oen_nen		timer7					gpio2[3]
9	T6	TIMER5	gpmc_be0_n_ce		timer5					gpio2[5]
10	U6	TIMER6	gpmc_wen		timer6					gpio2[4]
11	R12	GPIO1_13	gpmc_ad13	lcd_data18	mmc1_dat5	mmc2_dat1	eQEP2B_in	pr1_pru0.pru.r30_16	pr1[11][3]	
12	T12	GPIO1_12	gpmc_ad12	lcd_data19	mmc1_dat4	mmc2_dat0	Egcp2a_in	pr1_pru0.pru.r30_14	pr1[11][2]	
13	T10	EHRPWM2B	gpmc_ad9	lcd_data22	mmc1_dat1	mmc2_dat5	eHrppwm2B		pr1[0][23]	
14	T11	GPIO0_26	gpmc_ad10	lcd_data21	mmc1_dat2	mmc2_dat6	eHrppwm2_trigzone_in		pr1[0][26]	
15	U13	GPIO1_15	gpmc_ad15	lcd_data16	mmc1_dat7	mmc2_dat3	eQEP2_strobe	pr1_pru0.pru.r31_15	pr1[1][15]	
16	V13	GPIO1_14	gpmc_ad14	lcd_data17	mmc1_dat6	mmc2_dat2	eQEP2_index	pr1_pru0.pru.r31_14	pr1[1][14]	
17	U12	GPIO0_27	gpmc_ad11	lcd_data20	mmc1_dat3	mmc2_dat7	eHrppwm0_sync0		pr1[0][27]	
18	V12	GPIO2_1	gpmc_clk_mux0	lcd_memory_clk	gpmc_wait1	mmc2_clk			mcasp0_fsr	gpio2[11]
19	U10	EHRPWM2A	gpmc_ad8	lcd_data23	mmc1_dat0	mmc2_dat4	eHrppwm2A		pr1[0][22]	
20	V9	GPIO1_31	gpmc_csn2	gpmc_be1n	mmc1_cmd			pr1_pru1.pru.r30_13	pr1[1][31]	
21	U9	GPIO1_30	gpmc_csn1	gpmc_clk	mmc1_clk			pr1_pru1.pru.r30_12	pr1[1][30]	
22	V8	GPIO1_5	gpmc_ad5	mmc1_dat5						gpio1[5]
23	U8	GPIO1_4	gpmc_ad4	mmc1_dat4						gpio1[4]
24	V7	GPIO1_1	gpmc_ad1	mmc1_dat1						gpio1[1]
25	U7	GPIO1_0	gpmc_ad0	mmc1_dat0						gpio1[0]
26	V6	GPIO1_29	gpmc_csn0							gpio1[29]
27	U5	GPIO2_22	lcd_vsync	gpmc_a8				pr1_pru1.pru.r30_8	pr1[1][22]	
28	V5	GPIO2_24	lcd_pclk	gpmc_a10				pr1_pru1.pru.r30_10	pr1[1][24]	
29	R5	GPIO2_23	lcd_hsync	gpmc_a9				pr1_pru1.pru.r30_9	pr1[1][23]	
30	R6	GPIO2_25	lcd_ac_bias_en	gpmc_a11						gpio2[25]
31	V4	UART5_CTSN	lcd_data14	gpmc_a18	eQEP1_index	mcasp0_axr1	uart5_nrd		uart5_ctsn	gpio1[10]
32	T5	UART5_RTSN	lcd_data15	gpmc_a19	eQEP1_strobe	mcasp0_ahclkx	mcasp0_axr3		uart5_rsn	gpio1[11]
33	V3	UART4_CTSN	lcd_data13	gpmc_a17	eQEP1B_in	mcasp0_fsr	mcasp0_axr3		uart4_rsn	gpio1[9]
34	U4	UART3_RTSN	lcd_data11	gpmc_a15	eHrppwm1B	mcasp0_ahclkx	mcasp0_axr2		uart3_rsn	gpio2[17]
35	V2	UART4_CTSN	lcd_data12	gpmc_a16	eQEP1A_in	mcasp0_ahclkx	mcasp0_axr2		uart4_ctsn	gpio1[8]
36	U3	UART3_CTSN	lcd_data10	gpmc_a14	eHrppwm1A	mcasp0_axr0			uart3_ctsn	gpio2[16]
37	U1	UART5_RXD	lcd_data9	gpmc_a12	eHrppwm1_trigzone_in	mcasp0_ahclkx	uart5_brd		uart2_ctsn	gpio2[14]
38	U2	UART5_RXD	lcd_data8	gpmc_a13	eHrppwm1_sync0	mcasp0_ahclkx	uart5_nd		uart2_rsn	gpio2[15]
39	T3	GPIO2_12	lcd_data6	gpmc_a6	eQEP2_index			pr1_pru1.pru.r30_6	pr1[1][12]	
40	T4	GPIO2_13	lcd_data7	gpmc_a7	eQEP2_strobe			pr1_pru1.pru.r30_7	pr1[1][13]	
41	T1	GPIO2_10	lcd_data4	gpmc_a4	eQEP2A_in			pr1_pru1.pru.r30_4	pr1[1][10]	
42	T2	GPIO2_11	lcd_data5	gpmc_a5	eQEP2B_in			pr1_pru1.pru.r30_5	pr1[1][11]	
43	P3	GPIO2_8	lcd_data2	gpmc_a2	eHrppwm2_trigzone_in			pr1_pru1.pru.r30_2	pr1[1][8]	
44	P4	GPIO2_9	lcd_data3	gpmc_a3	eHrppwm0_sync0			pr1_pru1.pru.r30_3	pr1[1][9]	
45	R1	GPIO2_8	lcd_data0	gpmc_a0	eHrppwm2A			pr1_pru1.pru.r30_0	pr1[1][6]	
46	P2	GPIO2_7	lcd_data1	gpmc_a1	eHrppwm2B			pr1_pru1.pru.r30_1	pr1[1][7]	gpio2[7]

Système de fichiers sysfs



- sysfs est un **système de fichier virtuel** fourni par le kernel

Obtention d'informations et configuration

- **Exportation d'informations** provenant de plusieurs sources
 - Sous-systèmes du kernel
 - Devices hardware
 - Drivers de périphériques

Contrôle des LEDs

- LEDs contrôlables à partir de fichiers dans `/sys/class/leds`

Plusieurs fichiers pour configurer et interroger les LEDs

- **Plusieurs étapes** à suivre pour allumer une LED

- Désactiver le trigger par défaut
- Définir la luminosité

```
$ cd /sys/class/leds/beaglebone:green:usr2
$ ls
brightness  device  max_brightness  power  subsystem  trigger
uevent
$ echo none > trigger
$ echo 255 > brightness
```



Pin GPIO

- Une **pin GPIO** possède deux états différents
 - Connectée à la masse en état LOW
 - Connectée à 3.3 V en état HIGH
- Pin connectée à rien du tout est dite **flottante**
- **Numérotation** différentes des pins par Linux

Numéro de GPIO[chip]_[pin] est $32 \times chip + pin$

Contrôle des pins GPIO

- Association d'un fichier pour chacune des pins

Ces fichiers se trouvent dans le dossier /sys/class/gpio

- Plusieurs étapes à suivre pour accéder à une pin

- Exporter la pin dans l'espace user
- Configurer la pin
- Lire ou écrire sa valeur

```
$ cd /sys/class/gpio  
$ echo 44 > export  
$ cd gpio44  
$ echo out > direction  
$ echo 1 > value  
$ echo 44 > unexport
```



Bus I²C (1)

- Bus I²C pour connecter des périphériques à la BBB

Ne nécessitant pas de vitesse de communication élevée

- Protocole série simple qui utilise deux *signal wires*

Développé par Philips Semiconductors (NXP)

- Configuration la plus simple en mode **maitre-esclaves**

Chaque esclave est identifié par une adresse unique sur 7 bits

Bus I²C (2)

■ Installation des outils I²C tools

<http://www.lm-sensors.org/wiki/I2CTools>

```
$ i2cdetect -l
i2c-0  i2c          OMAP I2C adapter      I2C adapter
i2c-1  i2c          OMAP I2C adapter      I2C adapter
$ i2cdetect -r 0
WARNING! This program can confuse your I2C bus, cause data loss
and
worse!
I will probe file /dev/i2c-0 using read byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] y
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:      -- -- -- -- -- -- -- -- -- -- -- -- -- --
10:      -- -- -- -- -- -- -- -- -- -- -- -- -- --
20:      -- -- -- -- UU -- -- -- -- -- -- -- --
30:      -- -- -- -- UU -- -- -- -- -- -- -- --
40:      -- -- -- -- -- -- -- -- -- -- -- -- --
50: UU -- -- -- -- -- -- -- -- -- -- -- -- --
60:      -- -- -- -- -- -- -- -- -- -- -- -- --
70: UU -- -- -- -- -- -- -- -- -- -- -- -- --
```



Bus I²C (3)

- Plusieurs devices utilisent l'I²C sur la BBB



- 0x24 Power Management Integrated Circuit (PMIC)
- 0x50 EEPROM
- 0x70 TDA998x (HDMI framer)

Valeur	Signification
--	Il n'y a rien à l'adresse
UU	Il peut y avoir quelque chose mais indisponible
Nombre hexa à 2 chiffres	Un device est sur l'adresse et prêt à réclamer un driver

Programmation de haut niveau

- La BBB peut se **programmer** dans plusieurs langages

Il suffit de savoir lire et écrire des fichiers texte

- Par exemple, librairie **Adafruit_BBIO** en Python

```
1 import Adafruit_BBIO.GPIO as GPIO
2 import time
3
4 GPIO.setup("P8_12", GPIO.OUT)
5
6 while True:
7     GPIO.output("P8_12", GPIO.HIGH)
8     time.sleep(1)
9     GPIO.output("P8_12", GPIO.LOW)
10    time.sleep(1)
```

Entrée analogique

- La BBB a sept pins pouvant servir d'**entrée analogique**

Données converties par l'ADC en valeurs entre 0 et 1.8 V

- **Plusieurs senseurs** qui renvoient des données analogiques

Proximité, luminosité, température, accélération, orientation...

```
1 import Adafruit_BBIO.ADC as ADC
2 import time
3
4 ADC.setup()
5
6 while True:
7     print(ADC.read("P9_33"))
8     time.sleep(0.5)
```

Sortie analogique

- La BBB a huit canaux PWM (*pulse-width modulation*)

Chaque canal est associé à une ou deux pins

- **Plusieurs senseurs** qui renvoient des données analogiques

Proximité, luminosité, température, accélération, orientation...

```
1 import Adafruit_BBIO.PWM as PWM
2 import time
3
4 PWM.start("P8_13", 0)
5
6 for i in range(0, 100):
7     PWM.set_duty_cycle("P8_13", float(i))
8     time.sleep(1)
9
10 PWM.stop("P8_13")
11 PWM.cleanup()
```

En JavaScript

- **BoneScript** est une librairie Node.js pour contrôler la BBB

Exploitation de la programmation par évènements de JS

- Code source disponible sur GitHub

<https://github.com/jadonk/bonescript>

```
1 var b = require('bonescript');
2
3 b.pinMode("P8_13", b.OUTPUT);
4 var state = b.LOW;
5 setInterval(toggle, 1000);
6
7 function toggle() {
8     state = state == b.LOW ? b.HIGH : b.LOW;
9     b.digitalWrite("P8_13", state);
10 }
```

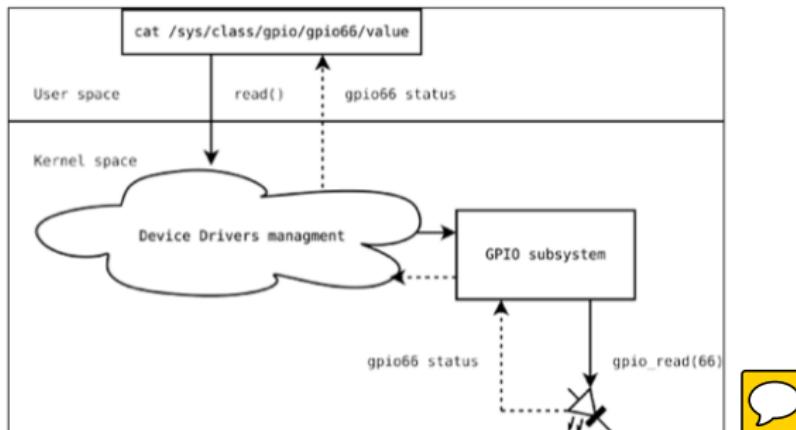
Driver de périphérique

- Contrôler un device hardware depuis la couche software

S'interface avec le hardware, expose une API dans le user-space

- Tout driver est un **fichier** sous Unix/Linux

Driver offre appels systèmes exécutables sur ce fichier



Types de périphérique

- Trois **types de périphériques** sous Unix/Linux
 - **Caractère** pour un accès par flux d'octets
Port série, périphérique audio...
 - **Bloc** pour hébergement d'un système de fichiers
Disque avec accès par blocs d'octets
 - **Réseau** pour gestion d'une transaction réseau
- Trois **types de drivers** associés aux types de périphériques
Par exemple /drivers/rtc, /drivers/pps (pulse per second)



Outil modutils

- Chargement module dans le kernel avec insmod et modprobe
La seconde commande charge toutes les dépendances
- Liste des modules chargés avec lsmod, détails avec modinfo
Nom, taille de octets et autres modules l'utilisant
- Déchargement d'un module avec rmmod

Exemple de driver Linux (1)



```
1 #include <linux/module.h>           /* Needed by all modules */
2 #include <linux/kernel.h>           /* Needed for KERN_INFO */
3 #include <linux/init.h>             /* Needed for the macros */
4
5 static int __init hello_start(void)
6 {
7     printk(KERN_INFO "Loading hello module...\n");
8     printk(KERN_INFO "Hello, World!\n");
9     return 0;
10 }
11
12 static void __exit hello_end(void)
13 {
14     printk(KERN_INFO "Goodbye Boris\n");
15 }
16
17 module_init(hello_start);
18 module_exit(hello_end);
19
20 MODULE_AUTHOR("Boris Houldroy");
```

Exemple de driver Linux (2)

```
bone# modinfo hello.ko
filename: /root/hello/hello.ko
srcversion: 87C6AEEED7791B4B90C3B50C
depends:
vermagic: 3.8.13-bone67 SMP mod_unload modversions ARMv7 thumb2
p2v8 bone# insmod hello.ko

bone# dmesg | tail -4
[419313.320052] bone-iio-helper helper.15: ready
[419313.322776] bone-capemgr bone_capemgr.9: slot #8: Applied #1
overlays.
[491540.999431] Loading hello module...
[491540.999476] Hello world
```

Driver versus librairie

- Un **driver** est exécuté dans le kernel space
 - Accès direct au hardware avec plus d'autorisations
 - Peut être contacté par des appels systèmes
- Une **librairie** est exécutée dans le user space
 - Exploite le hardware en faisant appel à des drivers
 - Profite de la sécurité offerte par l'OS



Crédits

- https://www.flickr.com/photos/bruce_aldrige/25151512373
- https://en.wikipedia.org/wiki/File:Desktop_personal_computer.jpg
- https://en.wikipedia.org/wiki/File:MSI_Laptop_computer.jpg
- https://en.wikipedia.org/wiki/File:Front_Z9_2094.jpg
- https://en.wikipedia.org/wiki/File:Blackberry_Z10.jpg
- https://en.wikipedia.org/wiki/File:ADSL_modem_router_internals_labeled.jpg
- <https://en.wikipedia.org/wiki/File:Debian-OpenLogo.svg>
- <https://en.wikipedia.org/wiki/File:Debian-OpenLogo.svg>
- https://en.wikipedia.org/wiki/File:Gentoo_Linux_logo_matte.svg
- <https://en.wikipedia.org/wiki/File:RedHat.svg>
- https://en.wikipedia.org/wiki/File:OpenSUSE_official-logo-color.svg
- https://en.wikipedia.org/wiki/File:Slackware_Logo_alt.jpg
- https://en.wikipedia.org/wiki/File:Arch_Linux_logo.svg
- https://en.wikipedia.org/wiki/File:Free_Software_Foundation_logo_and_wordmark.svg
- https://en.wikipedia.org/wiki/File:Openwrt_Logo.svg
- https://en.wikipedia.org/wiki/File:Openmoko_logo.png
- https://en.wikipedia.org/wiki/File:GeeXboX_Logo.png
- <https://www.flickr.com/photos/senimmi/5797414382>
- <https://www.flickr.com/photos/120586634@N05/14491195107>