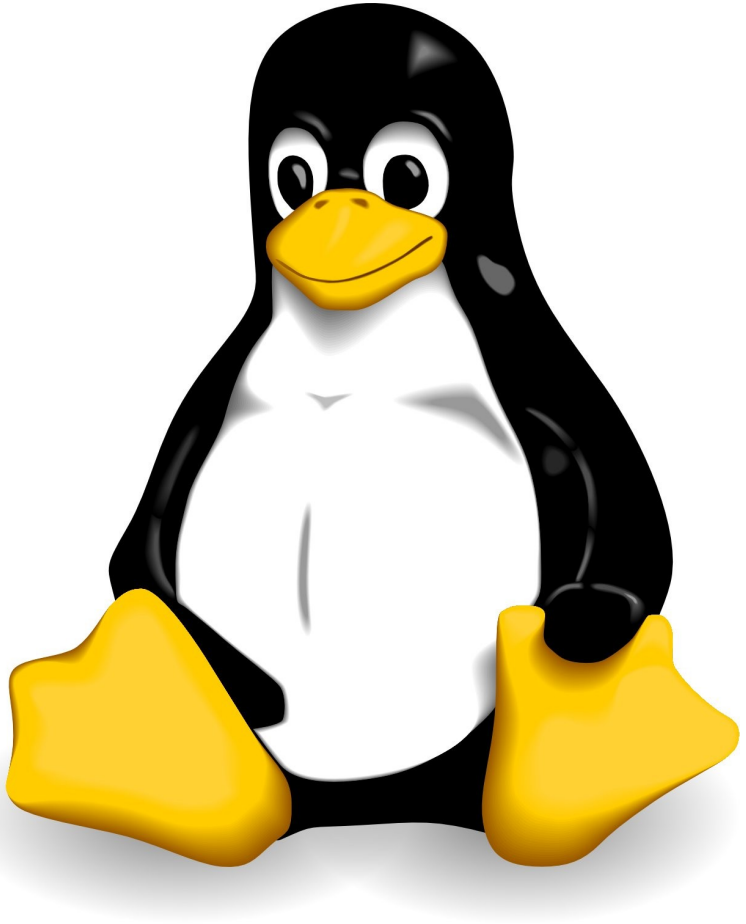
Atelier Linux Embarqué



Points abordés

- Présentation GNU Linux Embarqué
- Matériel
- Process de boot + kernel
- Le rootfs
- Cross-compilation
- Les modes
- Processus & Threads
- Signaux, pipes et IPC
- Kernel module

Présentation de Linux Embarqué

GNU/Linux

- 16/03/1953 : Naissance de Richard Matthew Stallman
- 28/12/1963 : Naissance de Linus Torvald
- 1971 : RMS entre au MIT, naissance du C et réécriture d'UNIX
- Septembre 1983 : Naissance du projet GNU
- Janvier 1985 : Sortie de l'OS GNU
  - Problème : pas de noyau libre
- Aout 1991 : Naissance du projet Linux
- Fevrier 1992 : Linux passe sous licence GNU
  - Naissance de l'OS GNU/Linux

Linux embarqué
Les outils/ distribution

- Builder une distribution embarqué :
  - Buildroot
  - Openwrt
  - OpenEmbedded
  - LFS
  - Gentoo
  - Yocto
- Utiliser une distribution pré-compilée :
  - Debian
  - Ubuntu
  - Fedora

Principales licences

- GPL
  - Plusieurs versions : V1, V2 et V3
  - Distribution obligatoire des sources
  - Caractère héréditaire
- LGPL
  - Enlève le caractère héréditaire de la GPL
- BSD
  - Aucune restriction sur la réutilisation des sources
- Creative Commons
  - Différentes déclinaisons
  - Déclinaisons proches de la GPL : BY et BY-SA

Le matériel

Architectures

- Les CISC :
  - X86/x86_64
  - PowerPC
- Les RISC :
  - MIPS
  - Sparc
  - SH
  - ARM

SoC ARM

- Histoire :
  - Acorn RISC Machine : 1983
- Avantages :
  - Consommation
  - Modulable
- Inconvénients :
  - Puissance

Famille ARM

- Source Wikipedia :

Architecture	Famille(s)
ARMv1	ARM1
ARMv2	ARM2, ARM3
ARMv3	ARM6, ARM7
ARMv4	StrongARM, ARM7TDMI, ARM8, ARM9TDMI
ARMv5	ARM7EJ, ARM9E, ARM10E, XScale, FA626TE, Feroceon, PJ1/Mohawk
ARMv6	ARM11
ARMv6-M	ARM Cortex-M0, ARM Cortex-M0+, ARM Cortex-M1 (voir ARM Cortex-M)
ARMv7-A	ARM Cortex-A (**ARM Cortex-A8**, ARM Cortex-A9 MPCore, ARM Cortex-A5 MPCore, ARM Cortex-A7 MPCore, ARM Cortex-A12 MPCore, ARM Cortex-A15 MPCore, Scorpion, Krait, PJ4/Sheeva, Swift
ARMv7-M	ARM Cortex-M (ARM Cortex-M3)
ARMv7-R	ARM Cortex-R (ARM Cortex-R4, ARM Cortex-R5, ARM Cortex-R7)
ARMv8	ARM Cortex-A50 (ARM Cortex-A53, ARM Cortex-A57), X-Gene, Denver

AllWinner A10 Cortex A8

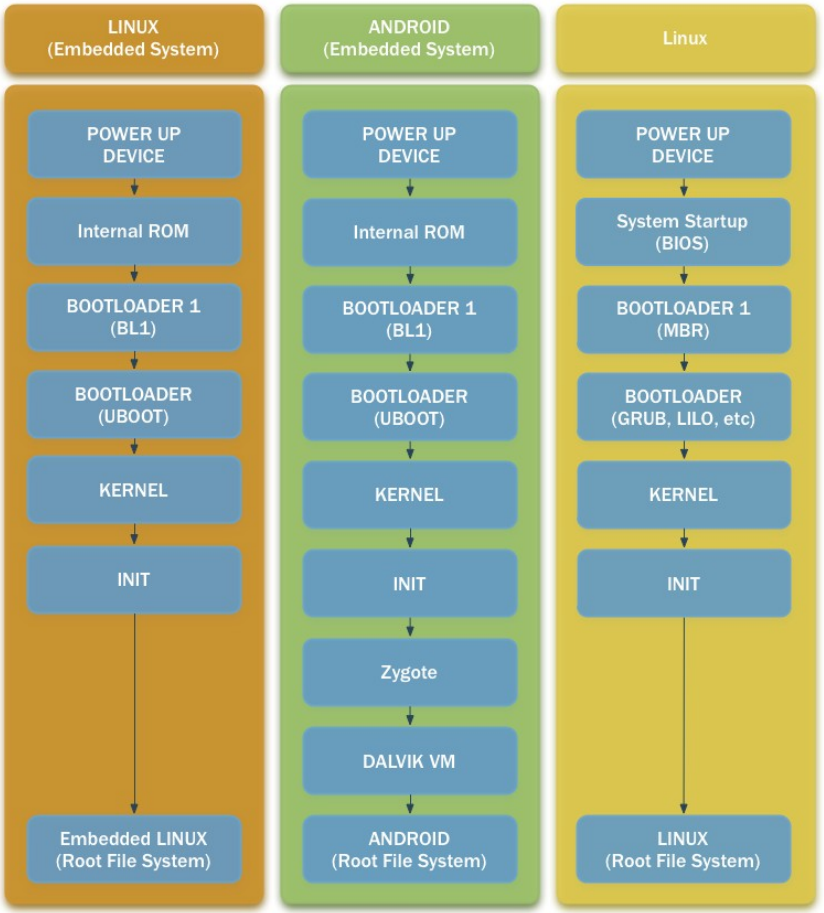
- 1.2ghz Cortex A8 ARM Core
- MALI400MP OpenGL ES 2.0 GPU
- DDR3 Controller 800MHz 1GB max
- 2160p Hardware-accelerated Video playback (4x the resolution of 1080p)
- 2D Accelerated Graphics (G2D) Engine
- a NAND Flash Controller that is capable of 8-way concurrent DMA (8 NAND ICs)
- 4 SDIO interfaces (SD 3.0, UHI class)
- USB 2.0 Host as well as a 2nd USB-OTG Interface (USB-OTG can be reconfigured as USB 2.0 Host, automatically)
- 24-pin RGB/TTL as well as simultaneous HDMI out
- SATA-II 3gb/sec
- 10/100 Ethernet (MII compatible)
- a 2nd 24-pin RGB/TTL interface that is multiplexed (shared) on the same pins for a standard IDE (PATA) interface.
- GPIO, I2C, PWM, Keyboard Matrix (8x8), built-in Resistive Touchscreen Controller, and much more.
- Web page : http://rhombus-tech.net/allwinner_a10/

pcDuino Board

- A Mini PC with Arduino type Interface powered by ARM Pro Spec:
  - CPU: 1GHz ARM Cortex A8
  - GPU: OpenGL ES2.0, OpenVG 1.1 Mali 400 core
- DRAM: 1GB
- Onboard Storage: 2GB Flash, SD card slot for up to 32GB
- Video Output: HDMI
- OS: Linux + Android
- Extension Interface: 2.54 mm Headers compatible with Arduino
- Network interface: RJ45 and USB WiFi Dongle

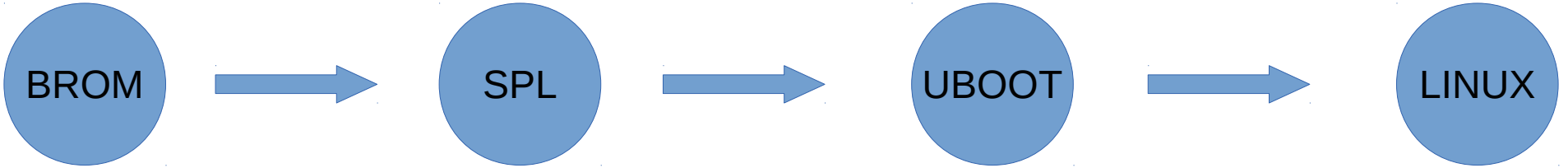
Process de boot

Le boot



- Source : http://javigon.com

Le boot



- Brom : Bootable ROM
- SPL : Secondary Program Loader
- U-Boot : Universal Bootloader
- Kernel Linux

Das U-Boot

- GNU GPL
- Très répandu
- Multi architecture : x86, ARM, MIPS, PPC, etc...
- Multi-board
- Shell interactif
- Téléchargement série ou ethernet
- Gestion de la mémoire et des disques

Quelques commandes u-boot

- help : liste toutes les commandes
- help <command> : aide de <command>
- fatload : charge un binaire en RAM
- printenv : affiche les variables d'environnement
- saveenv : sauvegarde les variables d'environnement
- run : lance la commande d'une variable

uImage

- Contient l'image du kernel
- Accompagnée d'un header :
  - Target Operating System
  - Target CPU Architecture
  - Compression Type
  - Load Address
  - Entry point
  - Image Name
  - Image Timestamp

Initramfs

- Prepare le boot du systeme
  - Charge les modules nécessaires au système
  - Monte les filesystems
  - Demande l'authentification pour les fs cryptés
- « Se lance » à la place de /sbin/init
- Mini rootfs compressé
- Busybox

Le kernel Linux

- C'est le cœur du système
- Interface entre le matériel et l'utilisateur
- Monolithique (un binaire) modulaire
- Multi-tâches
- Multi-utilisateur
- POSIX

Le ROOTFS

Le rootfs

- Root filesystem
- C'est l'ensemble des fichiers et répertoires du système
- Il contient au minimum les répertoires :
  - /dev, /proc, /bin, /etc, /lib, /usr, /tmp
- Plusieurs manières de le construire
  - From scratch
  - Via des utilitaires (debootstrap)

Créer un rootfs avec debootstrap

- Install un rootfs debian/ubuntu minimal dans un repertoire
- Le rootfs créé est directement utilisable
- La commande chroot permet de changer de racine ou de lancer une commande dans le nouveau rootfs

qemu

- Emulateur d'architecture
- Utiliser par des hyperviseurs comme kvm
- Multi-plateforme
  - Linux
  - Mac OS X
  - Windows
  - FreeBSD, OpenBSD, NetBSD

qemu-debootstrap

- Regroupe les avantages de 3 entités logicielles
  - debootstrap
  - qemu
  - module binfmt
- Permet de créer un rootfs complet pour toute architecture supportée par debian/ubuntu

CROSS-COMPILATION

cross-compilation

- Cross-compiler, c'est compiler pour une cible autre que celle que vous utilisez
- Elle permet d'utiliser les hautes performances d'un PC pour compiler plus rapidement une application pour la cible voulue.
- Pour cross-compiler, il est nécessaire d'utiliser une toolchain dédié à la cible.

Toolchain

- Permet de compiler
- Une toolchain comprend :
  - binutils : linker + assembleur
  - linux-headers
  - gcc : Compilateur C
  - glibc : Bibliothèque C du système

Les pages de man

1. Commandes utilisateur
2. Appels système
3. Fonctions de bibliothèque
4. Fichiers spéciaux
5. Formats de fichier
6. Jeux
7. Divers
8. Administration système
9. Interface du noyau Linux

Les MODES

Mode kernel

- C'est le mode du noyau Linux
- Accès sans restriction au matériel
- Toute la mémoire est accessible
- Zone de code à haut risque
- Conséquence d'un crash en mode kernel:
  - Freeze du système
  - Peut rendre le hardware inutilisable

Mode user

- C'est le mode applicatif
- Pas d'accès direct au matériel
- Pas d'accès direct à la mémoire
- Risque limité au périmètre du processus
- Crash en mode user :
  - Crash de l'application
  - Systeme toujours opérationnel

Interactions mode kernel – mode user

- Problématique : Comment accéder au matériel depuis l'espace utilisateur ?
- Solution : Le kernel met à disposition des appels système
- Appels système masqués dans la libc
- API utilisable dans tout code C
- Les accès au matériel sont donc sécurisés
- Certains appels système requièrent des droits root
- Les lister tous : man -s 2 -k .

Quelques appels système sur les fichiers

- access : vérification des droits sur un fichier
- chdir : changer le repertoire courant
- chmod : changer les permissions d'un fichier
- chown : changer le propriétaire d'un fichier
- open : ouvrir un fichier
- close : fermer un fichier
- read : lire un fichier
- write : ecrire dans un fichier

Quelques appels système sur les processus

- exit : quitte le processus courant
- fork : crée un processus fils
- getuid : Retourne l'id du user
- getgid : Retourne l'id du groupe
- getpid : Retourne l'id du processus
- getppid : Retourne l'id du processus parent

Appels systeme plus particuliers

- chroot : Change le répertoire racine
- inotify_init : Initialise un inotify.
- inotify_add_watch : Ajoute un watch sur un fichier
- kill : envoie un signal à un processus
- signal : gestion des signaux reçus
- ioctl : Input/Output Control

PROCESSUS
&
THREADS

Les processus

- Toute exécution d'un code sous Linux est représenté par un processus
- Chaque processus a un identifiant : le PID
- Chaque processus a un processus père : PPID
- Le processus init est le processus originel :
  - Il est lancé par le noyau
  - Il porte le PID 1

Les processus

- Chaque processus possède un ou plusieurs segments mémoire qui lui sont réservés.
- Un processus peut créer plusieurs autres processus
- Un processus peut se dupliquer : fork()
- Un processus peut interagir avec d'autres processus via :
  - les IPC
  - les signaux
  - les tubes

ETUDE DE CAS 01

Les Threads

- Les threads sont également une solution de parallélisation des taches.
- A la différence des processus :
  - Les threads partagent le même espace mémoire
  - pas de switch de contexte
  - Sont partagées :
    - Variables globales
    - Variables statiques
    - File descriptors

Les threads

- Creation d'un thread :
  - int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);
- Attente de la fin d'un thread :
  - int pthread_join(pthread_t thread, void **retval);
- Verrouillage/deverouillage d'un thread :
  - int pthread_mutex_lock(pthread_mutex_t *mutex);
  - int pthread_mutex_unlock(pthread_mutex_t *mutex);

ETUDE DE CAS 02

SIGNAUX, PIPES, & IPC

Signaux

- Pour les lister : kill -l
- Un signal permet à un processus d'interrompre un autre processus
- Les signaux sont un mécanisme asynchrone
- Un processus peut traiter un signal de 3 facons :
  - block : il est mis attente
  - Ignore : simplement ignoré
  - Catch : il est intercepté et traité par le processus

Liste des signaux POSIX man 7 signal

First the signals described in the original POSIX.1-1990 standard.

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from alarm(2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at tty
SIGTTIN	21,21,26	Stop	tty input for background process
SIGTTOU	22,22,27	Stop	tty output for background process

Gérer les signaux recus

- L'appel système signal() permet de changer l'action par défaut d'un signal reçu
- Mais signal() a deux inconvénients :
  - Il ne bloque pas les autres signaux
  - Il reset l'action à sa valeur par défaut
- Il vaut mieux utiliser l'appel système sigaction()

ETUDE DE CAS 03

Les pipes

- Ils sont un premier mécanisme de communication inter-processus
- Deux types de pipes :
  - les pipes classiques
  - Les named pipes (FIFO)

Les pipes classiques

- Communication entre deux processus
- La communication est unidirectionnelle
- Les file descriptors sont partagés lors d'un fork
- Les pipes permettent donc une communication d'un processus père avec sa descendance

Utilisation d'un pipe classique

- Création d'un pipe :
  - int pipe(int pipefd[2]);
- Lecture/Ecriture dans un pipe :
  - Toute fonction de lecture/ecriture de fichiers
    - ssize_t read(int fd, void *buf, size_t count);
    - ssize_t write(int fd, const void *buf, size_t count);
- Fermeture d'un pipe :
  - int close(int fd);
  - Attention à bien fermer les deux extrémités du pipe

Les named pipes

- Communication entre deux processus
- La communication reste unidirectionnelle
- Communication entre processus sans liens de parenté

Utilisation des named pipes

- Creation :
  - int mkfifo(const char *pathname, mode_t mode);
- Lecture/ecriture :
  - Toute fonction de lecture/ecriture de fichiers
    - ssize_t read(int fd, void *buf, size_t count);
    - ssize_t write(int fd, const void *buf, size_t count);
- Suppression :
  - int remove(const char *pathname);
  -

ETUDE DE CAS 04

Les IPC SystemV

- Inter Processus Communication
- Il existe 3 type d'IPC :
  - Les sémaphores
  - Les segments de mémoire partagée (shm)
  - Les queues de message
- Mécanisme synchrone
- Une commande pour les lister : ipcs

Les semaphores

- Les Sémaphores de Dijkstra (1965)
- Trois actions sur un sémaphore :
  - Init()
  - P()
  - V()
- Les sémaphores permettent de résoudre les problèmes d'accès concurrentiels

Utilisation des sémaphores

- Creation d'un semaphore :
  - int semget(key_t key, int nsems, int semflg);
- Initialisation d'un semaphore :
  - int semctl(int semid, int semnum, int cmd, ...);
- Opération sur le sémaphore :
  - int semop(int semid, struct sembuf *sops, unsigned nsops);

Les shm

- Segments de mémoire partagée
- Permet de partager un espace mémoire entre processus
- Si besoin, il doit être protégé des accès concurrentiels :
  - Utilisation des sémaphores (mutex)

Utilisation des shm

- Création/récupération du shm :
  - int shmget(key_t key, size_t size, int shmflg);
- Attache le shm au segment du processus :
  - void *shmat(int shmid, const void *shmaddr, int shmflg);

Message queue

- Permet l'échange de message entre processus
- Les messages peuvent être typés
- Selection des messages par type possible

Utilisation des message queues

- Creation de la queue :
  - int msgget(key_t key, int msgflg);
- Envoi de messages :
  - int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
- Reception de messages :
  - ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);
- Info, stats, suppression :
  - int msgctl(int msqid, int cmd, struct msqid_ds *buf);

ETUDE DE CAS 05

ioctl

- De plus en plus de drivers sont développés
- Il y a donc besoin de plus d'appels système
- Mais le nombre d'appels systèmes du kernel est limité.
- Solution : utiliser une fonction générique sur le device avec un flag et un paramètre spécifique au driver

Exemple ioctl I2C/SPI sur pcduino

- opResult = ioctl(i2cHandle, I2C_TENBIT, 0);
- opResult = ioctl(i2cHandle, I2C_SLAVE, 0x10);
- opRseult = ioctl(spiDev, SPI_IOC_RD_MODE, &mode);
- opRseult = ioctl(spiDev, SPI_IOC_WR_MODE, &mode);
- opResult = ioctl(spiDev, SPI_IOC_RD_MAX_SPEED_HZ, &maxSpeed);

KERNEL MODULE

Kernel module

- Aucune utilisation de librairies du user space
- Includes obligatoires :
  - linux/kernel.h
  - linux/module.h
  - linux/init,h
- Fonctions obligatoires :
  - module_init(x);
  - module_exit(x);

ETUDE DE CAS 06

Liens utiles

- pcduino :
  - http://www.pcduino.com/
  - https://github.com/pcduino/kernel
  - https://github.com/sparkfun/pcDuino
- U-Boot : http://www.denx.de/wiki/U-Boot
- API POSIX pour les threads : https://computing.llnl.gov/tutorials/pthreads/
- Outil de communication série : http://doc.ubuntu-fr.org/gtkterm
- qemu-debootstrap : https://wiki.ubuntu.com/ARM/RootfsFromScratch/QemuDebootstrap