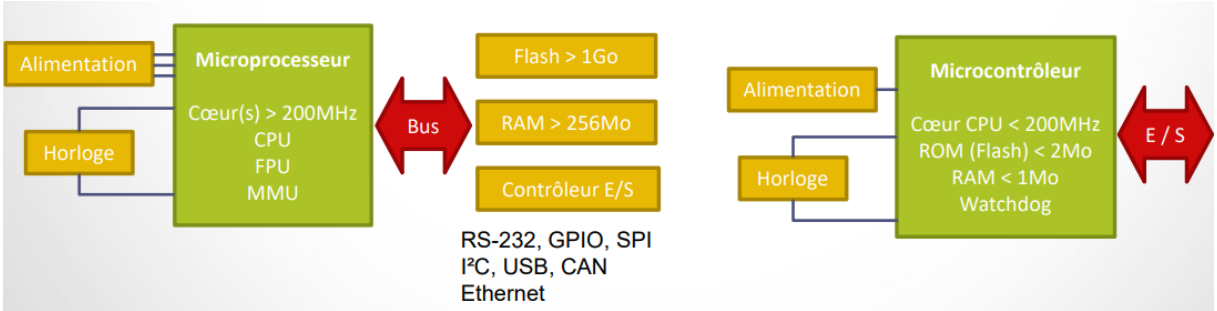
Systèmes
- Business critical -> time-to-market (telephone, audio, tv…)
- Mission critical -> qualité supérieur (imagerie, transmission…)

Systèmes critiques
- Life critical -> validation et certification (pacemakers, robots chirurgie…)
- Safety critical -> validation et certification (pilotage frein abs, distribution electronique…)

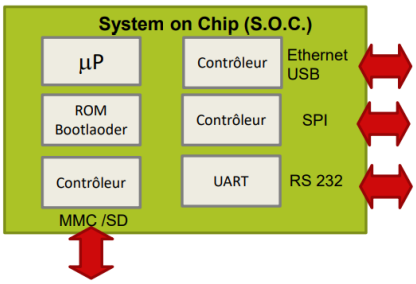
Processeur :
- Interprète instructions
- Traite données
- Nécessites éléments complémentaires :
  - Horloge
  - Mémoire pour exécuter programme (ram)
  - Mémoire stockage (rom)
  - Périphériques
- Utilise des bus
  - Adresses
  - Contrôle
  - Données

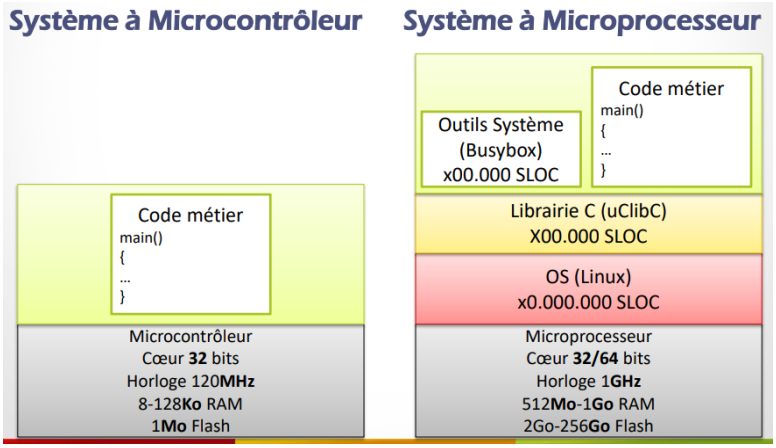
Microprocesseur :
- Intégration dans circuits distincts
- Nécessité interconnexion (bus, câblage)
- Plus gros
- Consomme et chauffe plus
- Plus cher
- Nécessite os
- Protection code métier difficile
- Ordinateur est un microprocesseur

Microcontrôleur :
- Tout sur un seul circuit
- Composant autonome, exécute programmes sur sa rom
- Plus petit
- Moins cher
- Moins de capacité que microprocesseur
- Souvent sans OS
- Protection code métier facile (griller un fusible et code inaccessible)
- Arduino est un microcontrôleur



System on chip
- Contrôleurs entrées sorties déjà présents
- Intégration électronique plus complexe
- Raspberry est un soc

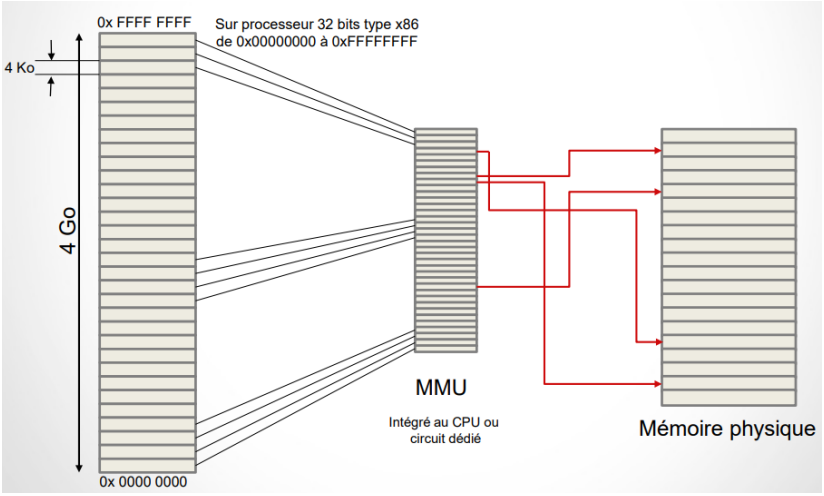




Gestion mémoire virtuelle

Adresses mémoires gérées dans processus ne sont pas directement références à adresses physiques

- Chaque processus a son espace d'adressage virtuel
    - Circuit MMU opère conversion via registre de tables de conversion
    - Lors changement contexte (= processus courant) charge registres MMU pour ce processus là



Critères comparaison systèmes embarqués :

- microprocesseur / microcontrôleur (classe processeur)
- MMU / sans MMU (gestion mémoire)
- OS / sans OS
- Temps normal / temps réel (type d'os)

OS : Ensemble programmes qui dirige utilisation des capacités de la machine

- Fait abstraction matériel
- Gere temps (partagé / normal)
- Gérer la distribution (entre processeurs, mémoires, périphériques)
- Appels système : read write open (servir processus)
- Déroutements : division 0, débordement pile (interruptions processus)
- Interruptions : clavier, souris, réseau (interruptions matériel)
- Assurer tâches système : swap, caches, pages (services)
- Composé de :
    - Noyau
    - Librairies
    - Programmes utilitaires

Noyau

Premier programme à s'exécuter après bios
Gestion processus, mémoire, fichiers, ipc, ordonnancement…
Linux stocké /vmlinuz ou /boot/vmlinuz
Implémente abstraction processus (mais ce n'en est pas un lui-même)
Fourni interface matériel
Types de noyau

- Monolithique
    - Fournit tous les services (programme unique, modulaire ou non
    - Tout s'exécute en mode noyau
        - Non modulaire : ancien windows, ancien mac os
        - Modulaire : linux > 1.2, bsd, solaris
    - Plus facile à écrire
    - Plus performant
- Micro
    - Fournit services minimaux (gestion processus, mémoire, ipcs)
    - Autres services sont fournis par des programmes utilisateurs
    - Plus résistant aux bugs
    - Plus difficile en pratique
- Hybride
    - Combinaison des deux
    - Windows 7, 10, iOS, Mac OS X

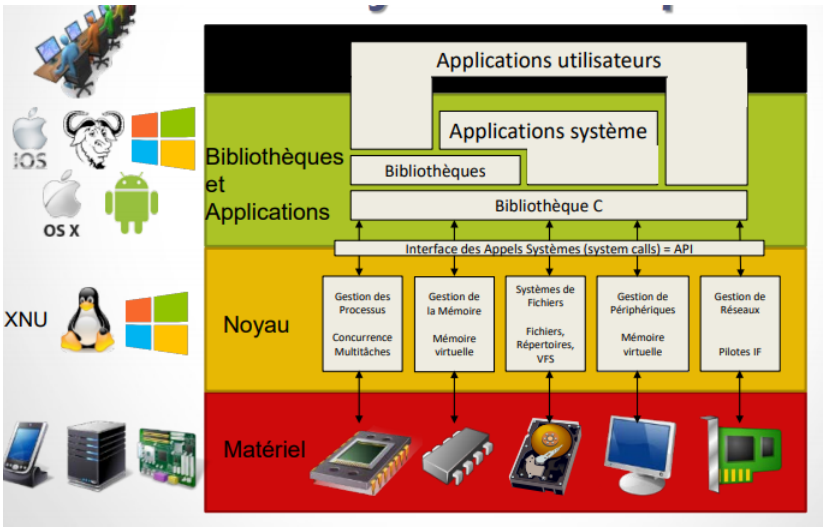
Librairies

- Librairie C, math…

Programmes utilitaires

- Programmes outils permettant manipulation du système via commandes basiques (shell, cp, rm…)

Architecture d'un os



Linux : noyau linux mais os gnu (bibliothèque C gnu lib C)

Compiler un noyau inutile pour pc standard car mises à jour régulières
Utile lors du développement du noyau, activer certaine fonctionnalité ou optimisé pour l'embarqué

Virtualisation

- Faire tourner plusieurs systèmes sur une seule machine
- Os hote (host)
- Os invité, virtualisé (guest)
- Couche d'abstraction matérielle / logicielle
- Partitionnement, isolation et partage des ressources
- Images manipulables (arrêt, gel, sauvegarde…)
- Types virtualisation
    - Isolation
        - Isole exécution des applis dans des contextes d'exécution
        - Performant mais pas réelle virtualisation
    - Noyau en espace user
        - Indépendance par rapport host
    - Machine virtuelle
        - Logiciel qui émule et ou virtualise matériel pour os guest
        - Cout en performances

Emulateur

- Permet exécution programme système X sur Y
- Instruction exécutée par routine qui simule pc
- Lent

Virtualisation complète

- Compile instructions lors de leur 1ère exécution
- Si possible code exécuté sur cpu sinon ré-écriture dynamique (80% de la vitesse)

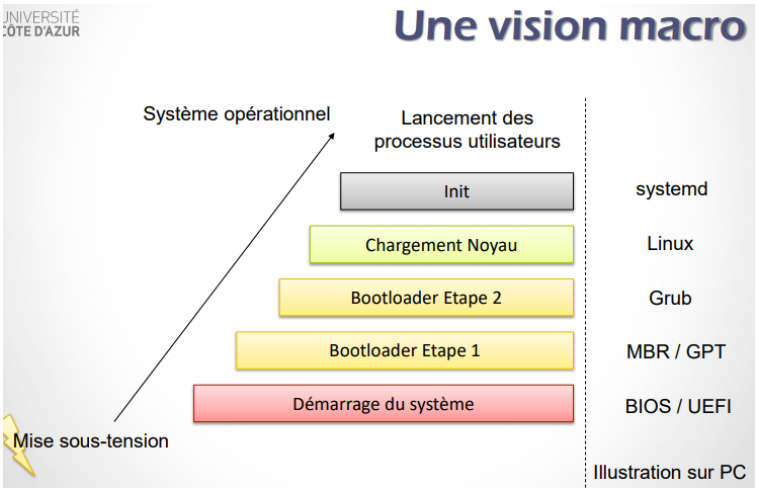
Para-virtualisation ou hyperviseur

- Système invité a conscience du système sous-jacent
- Performances optimales

Chargement OS : Chargement successif de plusieurs programmes

- Charger noyau compliqué et peut pas se faire en une opération
    - Trop volumineux
    - Programme chargement noyau est chargé avec un programme chargeur
- Périphérique ou trouver le noyau est paramétrable

Système de chargement est dépendant de la plateforme



Bios : Basic Input Output System

- Microprogramme
- Effectue opérations élémentaires lors de mise sous tension
- Développé en Assembleur
- 1 mo de mémoire accessible
- Master Boot Record -> Boot loader -> Kernel -> os

UEFI : Unified Extensible Firmware Interface

- Successeur bios
- Développé en C
- EFI Boot Loader -> Kernel -> os

BIOS: choix d'un périphérique d'amorcage

Partitionnement

- Subdivision de l'espace de stockage
- Au moins une
- Nécéssité d'une table de partition

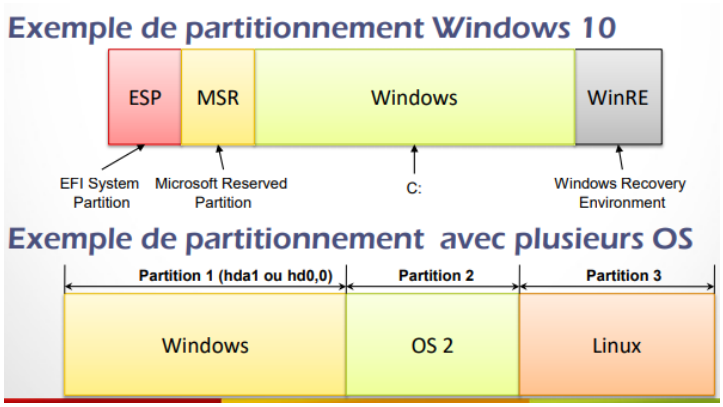


Table de partitionnement

- MBR : Master Boot Record
  - Lance bootloader
  - Table de partitionnement
  - BIOS et UEFI
  - 4 partitions max, < 2.2 To chacune
- GPT : GUID Partition Table
  - MBR protecteur
  - GPT primaire : entête
  - GPT secondaire : table de partitionnement
  - UEFI
  - 128 partitions max, < 256 To chacune

Séquence de boot

Chaque processeur s'initialise

- Multiprocesseurs : élection d'un cpu leader

CPU leader exécute instructions en 0xfffffff0

- Saut vers début du programme bios

BIOS : POST (power on self test)

Exemple MBR et BIOS:

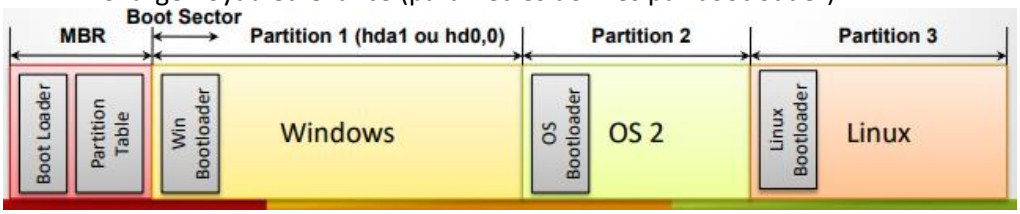
Charge MBR du périphérique de boot

Inspection MBR

- Verification (table des partitions)
- Recherche secteur de boot

Charge secteur de boot

- Charge programme chargeur (Bootloader) (Lilo (plus utilisé), Grub, Windows…)
- Charge noyau et le lance (paramètres donnés par bootloader)



Configuration de grub

- /boot/grub/grub.cfg
- Définition possibilités de démarrage
- Passage de paramètres au noyau

```
menuentry "Debian GNU/Linux" {
        set root=(hd0,3)
        linux /vmlinuz root=/dev/hda3 ro
        initrd /initrd.img
}
```

set root=(hd0,3)

- utilise 3eme partition du premier disque

linux /vmlinuz root=/dev/hda3 ro

- indique chemin noyau /vmlinuz puis paramètres

initrd /initrd.img

- utiliser l'image initrd/initramfs située en /initrd.img

Exemple de paramètres

Avec valeur :

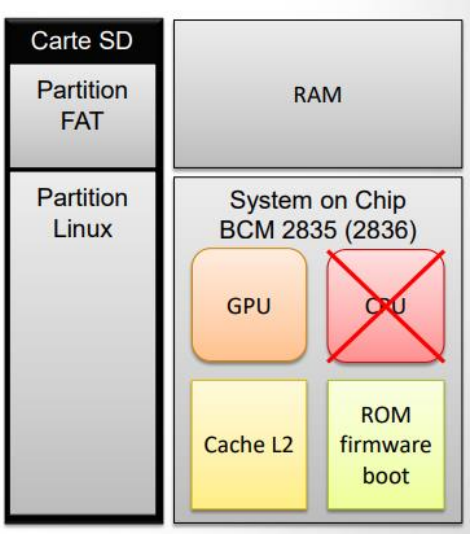
- init= : défini exécutable à lancer après montage racine
- root= : défini partition racine à monter via
    - UUID
    - /dev/…

Sans valeur :

- Ro ou rw : defini partition racine lecture ou ecriture
- Quiet : ne pas afficher messages au démarrage
- single : démarrage en mode single user

Démarrage Raspberry Pi

GPU : milliers de cœurs pour traiter efficacement tâches simultanées
CPU : nombre restreint de cœurs optimisés pour un traitement en série



Mise sous tension
GPU activé, CPU en veille, carte SD désactivée
GPU exécute programme en ROM
    Premier étage bootloader
Permet accès Carte SD
Accès première partition en FAT
Chargement bootcode.bin dans cache et exécution par gpu
Met en service ram
GPU exécute start.elf
Charge config.txt et cmdline.txt
Sépare mémoire en deux zone
    Ram gpu et ram cpu
Chargement noyau
Exécution noyau par cpu
Montage partition racine
Lancement du premier processus

Les bootloader sont adaptés à la plate-forme, embarqué, ordi bureau…

Démarrage des services

- 1eres instructions après boot décompression noyau
- Exécution noyau (perifs, monte racine, lance premier processus pid 1)
- Lance script de service :
    - System V Init : 1983
    - Systemd : 2010 remplace system v init
- Monte systèmes de fichiers
- Démarrage services, pilotes…

System V init

Processus init

- Lit fichier /etc/inittab
- Exécute scripts rc (Run Control)
- Se trouve dans un état (niveau) d'exécution : runlevel
- Scripts démarrage de service : /etc/init.d
- Quand init change de niveau tâches sont déclenchées
    - En fonction des règles dans le fichier /etc/inittab
    - En onction des scripts de Run Control des répertoires /etc/ren.d

Mécanisme simple, basé sur fichiers textes et scripts shell mais démarrage séquentiel des scripts donc lent et non opti

Systemd : system deamon
Déplace problème de démarrage des services à la gestion du system
Organisation des processus en groupes de contrôle (cgroup)

- Supervision des services
- Controle des services et de leurs environnements
- Chaque tâche est une unité ayant un type

Les cibles (targets) sont des groupes d'unités
Quand systemd lancé par noyau lit fichier /etc/systemd/system.conf
Efficace car processus en parallèle mais plus complexe que sysvinit

Logiciel libre : signifie liberté et non gratuité
Licence copyleft : modification doivent être publiques
Licence non-copyleft : modifications peuvent rester propriétaires

Bibliothèque : fichier unique regroupant un ensemble de fichiers objets précompilés

- ar : crée une bibliothèque regroupant des .o
- ranlib : ajoute un index à la biblio
- ld : crée une biblio partagée
- statique
    - Unix / MacOs : .a
    - Windows : .lib
    - Inclus dans l'exécutable à la compilation donc duplication dans fichier et mémoire
    - Pour la créer :
        - Compilation .c en .o : gcc –c fichier1.c fichierX.c
        - ar c libmylib.a fichier1.o fichierX.o
        - ranlib libmylib.a
- Partagée (dynamique)
    - Unix : .so
    - Windows : .dll
    - MacOs : .dylib
    - Evite duplication code dans exécutables
    - Edition de lien dynamique (chargement à la volée) : coût en perf
    - Pour la créer :
        - Compilation .c en .o : gcc –fPIC –c fichierX.c
        - Gcc –shared –Wl,-soname,libmylib.so.3 mylib.so.3.2 fichierX.o

Ldd cheminProgramme : permet de savoir lib dynamiques utilisées

Bibliothèque C réalisée par projet GNU : glibc

- Faite pour les performanes et respect standards
- Machines de bureau ou serveurs

Bibliothèque c pour embarqué : uClibc

- 4 fois plus petit que glibc (600ko vs 2.5mo)
- Fortement compatible
- Systèmes à faible mémoire

Les autres :

- Adaptées pour tous petits sytèmes, des init, ramdisks ou initramfss

BusyBox : regroupe la plupart des utilitaires Unix en un exécutable

- Compilé statiquement uClibc : 500ko
- Compilé statiquement glibc : 2Mo
- Fournit implémentation de init dans /etc/inittab
- Make defconfig : pour débuter avec busybox
- Make allnoconfig : déselectionne toutes options, bien pour inclure seulement ce qu'on veut
- Compilation : make
- Installation : make install
    - Arborescence crée dans dossier _install
    - Cp –a _install/* /mnt/

Jeux d'instructions

CISC : Complex Instruction Set Computer

- Faire tâche complexe en minimum instructions assembleur
- Mise sur matériel pour faire tâches de plus en plus complexe

RISC : Reduced Instruction Set Computer

- Jeu d'instructions simple qui peuvent être effectuée en un cycle
- Mise sur logiciel pour enchainer vite les instructions

Endianness : ordre dans lequel les octets sont rangés en mémoire, soit nombre 0xA0B70708

Big-endian : octet poids fort dans adresse la plus petite (A0)
Little-endian : octet poids faible dans adresse mémoire la plus petite (08)
Bi-endian : supporte les deux

Compilation croisée : produire du code pour un autre processeur

- Plus rapide/facile de travailler sur station de travail que sur cible

Etapes compilation

- Pré-processeur
    - Lit fichiers sources (.c .cpp..)
    - Traite directives
    - Produit fichier source ou directives ont été exécutées
- Compilateur
    - Traite source et le produit sans #
    - Transforme source en intermédiaire (.o…) compilé
- Editeur de liens
    - Prend fichiers objets et assemblent
    - Produit exécutable / librairie

Compilation croisée du noyau : en redéfinissant variables ARCH et CROSS_COMPILE

- ARCH=arm
- CROSS_COMPILE=arm-lib...
    - Dans makefile (non recommandé)
    - Dans ligke de commande make ARCH=…
    - Dans variables d'environnement

Puis ajouter chaine de compilation croisée au path

3 machines à distinguer pour création chaine compilation

- Machine de construction (build)
    - Machine ou est construite machine de compilation
- Machine hôte (host) (build et host souvent meme)
    - Machine sur laquelle sera exécutée compilation

- Machine cible (target)
    - Machine destination du binaire

Eléments d'une chaine de compilation croisée

- Compilateur (gcc pour linux embarqué, support c c++ java…)
- Binutils
    - As : transforme code généré en binaire
    - Ld : regroupe codes objets en librairies et executables
- Débugeur (optionnel)
- Librairie C
- Entêtes du noyau pour compiler libC

Optimisation

- Vitesse production pour cible
- Vitesse boot
    - Enlever message boot via option quiet (réduit 30 à 50% temps demarrage)
    - Mettre ce qui est pas nécessaire au boot en modules
    - Spécifier allocation de mémoire au boot
- Vitesse système
    - Remplacer commandes et utilitaires externes par busybox
    - Réduire commandes pipe
    - Paralléliser démarrage des services
- Vitesse applis
    - Make –o pour desactiver optimisations, ou les activer (5 niveaux dopti)
    - Executables statique pour gagner temps pas de lien dynamique
    - Executable librairies partagés : utiliser Pre linking
- Réduire taille : emprunte disque et mémoire
- Réduire consommation

Options pour une compilation plus rapide

- Pipe : Utilise tube au lieu de fichiers temp
- J : specifie nombre de jobs simultané

Faire des mesures

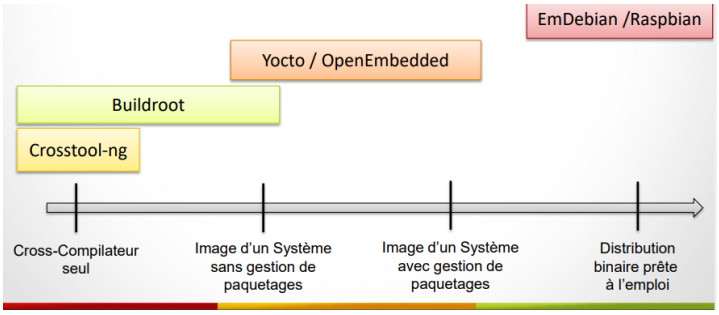
- CONFIG_PRINTK_TIME : ajoute des infos temporelles
- CONFIG_BOOT_TRACER : dans script noyau, mesure temps des initcalls
    - Pour voir info generer svg commande dmesg | perl
- Jiffy : unité mesure tres courte (arm 1 jiffy = 10ms)
    - Temps entre deux interruptions timer

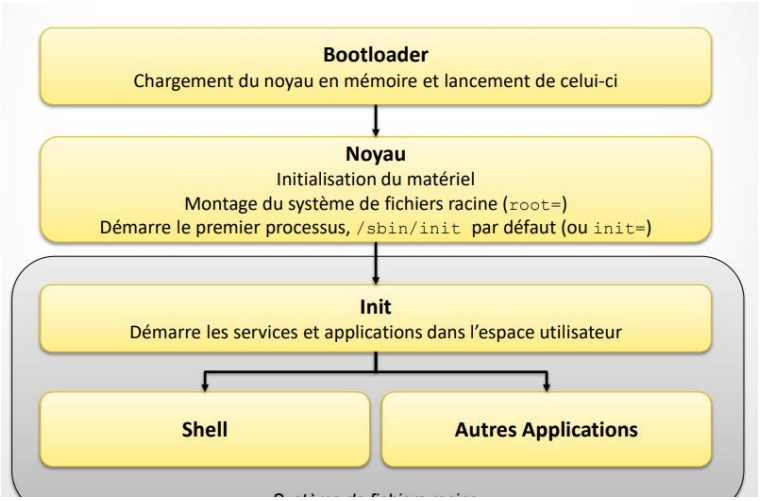
Reduire taille

- Reduire taille noyau
    - Linux tiny
- Librairies partagées
- Librairie C plus compacte
- Stripper exécutables (supprime des infos)
    - Commande strip ou sstrip

Economie d'énergie

- Powertop : montre les 10 sources de consommation
- Mesure nombre de wake_up et temps en low_power
- CPU_FREQ permet de changer fréquence du cpu a la volée





Distributions binaires pour embarqué

- Pas besoin de créer système from scratch
- Beaucoup d'applications et outils prêts à l'emploi
- Mise à jour facile
- Problèmes
    - Pas optimum en taille et perf (1go)
    - Difficile de supprimer fonctionnalités non souhaitées

Outils pour construction de system embarqués

- Buildroot : crée une image d'un système complet sans paquetage
    - Peut compiler plupart des applications, busybox, python…
- Yocto/OpenEmbedded…

Systèmes embarqués considérés

- Systèmes en temps réels
    - Soumis à des contraintes temporelles
    - Pas forcément rapide
    - Contraintes matérielles
        - Mémoire
        - Cout
- Systèmes critiques
    - Prédictibilité accrue
        - On veut pouvoir prédire comportement d'un système
        - Fonctionnel : comportement déterministe (meme entrée produit toujours meme sortie)
        - Temporel : calcul doit être fait avant des échéances

Méthodes de validation
- Tests
- Simulation fonctionnelle

Critères de validation
- Temporelle
- Enégétique
- Empreinte mémoire

Micro controleur sans os
- Os prend de la place
- Os complique conception
    - L'os prend la main ?
    - Combien de temps ?
    - Quand mon code est exécuté ?
    - Est il interrompu ?
- Os utilise t'il toute la puissance matérielle ? (sleep, power down, standby…)
- On dev nous meme un os
- Aucun os dispo
- Programmation mono tache
- Prédictibilité forte
- Optimisations possibles
- Gain en perfs
- Environnement de dev :
    - langage
    - compilater
    - linker (transfert vers machine)

Entrées sorties
- permet communication avec électronique
- multi fonctionnalités
- multi directionnelles

Registres
- Stockés en RAM
- 8 / 16 / 32 bits
- Adresse mémoire particulière

Stockage du programme
- RAM pour les premiers tests
- EEPROM (Flash) pour les tests plus poussés
- ROM en production

Stockage des variable : en RAM
Stockage pile : en RAM (fin ou début suivant uc)

Watchdog : dispositif matériel et logiciel contre plantage, qui reset le cpu pour relancer app si programme n'a pas fait signe à temps au watchdog

Stratégies d'implémentation
- Sans interruption (Synchrone)
    - Boucle infinie -> Lecture capteurs, calculs, écriture actionneurs

    - Avantages : Facile, calcul de temps de réactions simplifiés, peu de gestion de pile
    - Inconvénients : peu réactif, difficile de gérer consommation électrique
- Avec interruption (arrêt temporaire d'un programme pour exécuter un autre)
    - Utilise handler = routine d'interruption pour catch interruption
    - Avantages : Traitement urgence, Attente non active, consomme moins, réactif
    - Inconvénient : plus dur à prendre en main
    - Modes :
        - Ignore interruptions (programme recoient pas)
        - Masque interruptions (programme recoient mais on prend pas en compte)
        - Compte les interruptions et on les traite successivement (si possible par materiel)

AVR gcc : cross compilateur basé sur gcc
Linker : avrdude, librairies pour archi spécifiques (avr-libc)

Suivant compilateur handler différents
- AVR-gcc : ISR(PCINT2_vect){//code}
- ICCAVR : #pragma fonction_handler ISR :NUM fonction_handler{//code}

Types d'os
- RTOS (real time) : multi tache, actions déterministes et interruptibles
    - Not fair
- GPOS (normal general purpose) : multi tâche récent, prends main sur tâches en cours, non déterministe, non interruptible
- Kernel
- Software executive
- Scheduler

Tâche
- Identifiant
- Référence vers code
- Priorité
- Etat
    - Prêt : mémoire allouée
    - Bloqué : en attente de ressource / évènement
    - En cours : choisie par ordonnanceur pour s'exécuter
    - Suspendue : en mémoire mais sera pas ordonnancée
    - Morte : plus de mémoire allouée
- Contexte
    - Registre
    - Pile…
- Task :
    - Periode
    - Echeance
    - Execution Temps
    - Priorité

Semaphore : Booléan ou compteur
- Synchronisation tâches / Exclusion mutuelle
- Interruption logicielle
- Primitives
    - Creation

    - Destruction
    - Depot (incrementation)
    - Retrait (decrementation)

Priorités des tâches

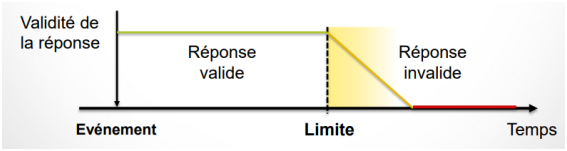
- RMA : Rate Monotonic Analysis, trouve priorités fixes aux taches, priorité haute aux traitements fréquents
- EDF : Earliest Deadline First, propose priorités dynamiques, priorité haute à tâche la plus proche de son échéance

Ordonnanceur

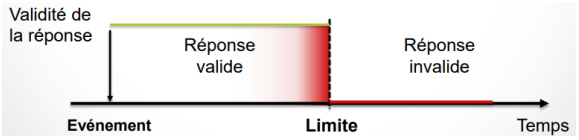
- Système ordonnancable si moyen d'exécuter toute tâches avant leur échéance
- Sans priorité non pré-emptif
    - Tient à jour liste tâches dans état prête
    - Lance exécution prochaine dès que tache en cours rend main
- Sans priorité pré-emptif
    - Tient à jour liste taches dans état prête
    - Lance exécution prochaine dès que tache en cours passe état bloquée suspendue ou morte
- Avec priorité pré-emptif
    - Tient à jour liste tâche dans état prête
    - Lance exécution prochaine tâche plus haute priorité dès que tâche plus haute est prête. Tâche en cours devient bloquée (si tache plus haute attends ressource plus basse, la basse devient prioritaire)

Temps réel : system (materiel et logiciel) satisfait contraintes fonctionnelles et temporelles imposées

- Si temps d'exécution < période de la tâche
- Mou/souple : satisfait échéances avec tolérance (lecteur mp3, video fps differents…)



- Dur/critique : satisfait échéances sans tolérance



- Simple : 1 seule tâche
    - Pas besoin os
    - Tache exécutée a chaque interruption d'horloge
- Complexe : plusieurs tâches en parallèle et periode differente
    - Pb de concurrence
    - Politique d'ordonnancement

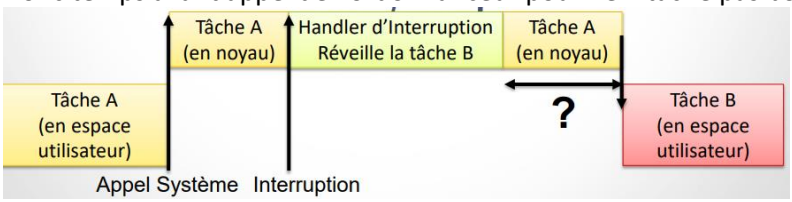
Autres systèmes temps réel

- Strict non certifiable : ordonnancement visant à approcher temps réel strict certifiable mais sans garantie
- Dur ou strict certifiable : nano secondes pourront intervenir dans temps de réponse mais prédictible et borné
- Absolu : temps parfaitement connu, identique stable et prédictible

Linux temps réel : linux pas temps réel de base

- Pour rendre temps reel

    - Modifier noyau (latence bornée + api de temps reel) : le projet PREEMPT_RT
        - CONFIG_PREEMPT_NONE : code noyau jamais préempté (comportement par défaut dans noyau standard)
        - CONFIG_PREEMPT_VOLUNTARY : Code noyau peut se préempter lui-même (pour appli desktop reactives)
        - CONFIG_PREEMPT_RT_FULL : plupart du code noyau peut être préempté à n'importe quel moment (pour systemes embarqués avec latence faible)
        - CONFIG_HIGH_RES_TIMERS : pour active timers haute resolution
    - Ou ajouter couche sous le noyau
- Linux est préemptif
    - Tache peut être exécutée dès retour interruption
    - Mais par défaut noyau pas préemptif, seulement les tâches
        - Donc temps avant appel de l'ordonnanceur pour new tache pas borné



    - Nombreux services de lib C ou noyau sont pas conçu avec préoccupation de temps

High resolution timers

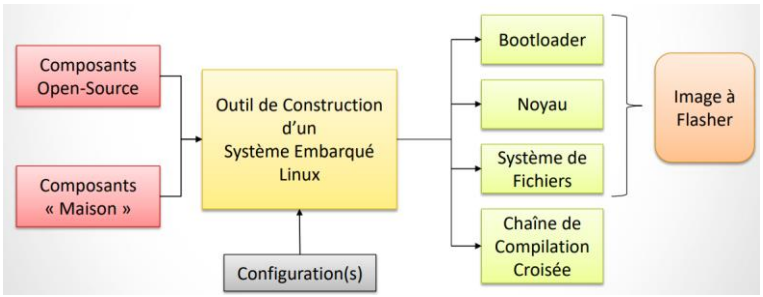
- Résolution des timers liés aux ticks régulier d'horloge
    - Augmenter fréquence tick pas bonne idée car consomme plus
- High resolution timers mis en place pour declencher interruption au bon moment
    - Un timer materiel gere plusieurs timers logiciels

Architecture mixte : microprocesseur + controleur

- Microcontroleur ajoute GPIO entrees analogiques, CAN
- Code uc verouillable fusible
- Possibilité surveiller cpu par microcontroleur
- Mais + cher
- Protocole communication entre les deux

Industrialisation

- Integration continue, gestion de version
- Utilisation boites à outils pour embarqué : busybox
- Utilisation systèmes de construction : buildroot, yocto
- Configuration du noyau optimisé (temps de boot, mémoire, drivers…)
- Mécanisme de déploiement et de mise à jour



Synthèse :
Controleur de vitesse d'un moteur : microcontroleur
Montre connectée : microcontroleur
Camera : microprocesseur
Sous marin caméra capteur : les deux

TD

TD1

Make -j 2*nb cœurs -> compiler avec plus de cœurs

Configurer noyau

- Make menuconfig : modifier configuration existante
- Make defconfig : configurer par défaut pour l'architecture
- Make oldconfig : validation et mise à jour d'une ancienne config
- Make allnoconfig : permet d'avoir configuration minimum
    - Environ 550ko bzimage
    - Inclus options nécessaires au fur et à mesure

Compilation noyau et modules : make
Installation du noyau : make install

Ajout noyau dans grub :

- cp /work/td02/linux-2.6.32.68/arch/x86/boot/bzImage /boot/vmlinuz-2.6.32.68
    - #bzImage a été obtenu après avoir compilé le noyau
- cp /work/td02/linux-2.6.32.68/System.map /boot/System.map-2.6.32.68
- cp /work/td02/linux-2.6.32.68/.config /boot/config-2.6.32.68

Dans boot/grub/grub.cfg enlever quiet sinon pas de message d'erreur

Pour installer patchs :

- cd /work/td02/linux-2.6.32.68
- patch -p1 /media/sf_shared_folder_vm_isle/LES_PATCHS

TD2

Busybox: contient lib C, code pour cp, ls…

- on compile : make install
- on copie fichiers busybox/_install dans root.img monté dans /mnt/
    - cp -aR _install/* /mnt/

Créer fichier avec block size : mke2fs –m 0 –i 1024 –F root.img
Trouver numéros majeurs et mineurs : ll /dev/console (ls -l)
Majeur correspond au driver utilisé dans noyau pour y accéder
Fichier dans /dev permet d'accéder à un periph virtuel ou physique
Créer un fichier character : mknod dev/console c 5 1 (majeur puis mineur)

TD5 optimisations

Clean installation (supprimer aussi .config) : make mrproper

- make clean supprime pas .config

On copie un nouveau .config a la place : make oldconfig pour verifier que .config fonctionne
On peut mettre quiet lors du boot pour pas afficher les messages

TD6 production de système embarque - raspberrypi

Donner type partition ; `mount | grep "^/dev"`

Partition 0 : VFAT

`cd /mnt/raspberry/loop0p0`

Fichiers dans la partition 0 servent

config.txt : spécifie le fichier noyau et assignation de variables, config du bios

cmdline.txt : passage de paramètres pour charger le noyau

`df -h loop0p0` : donne la taille de la partition et la taille réelle utilisée

- ex : taille 30mo mais taille reelle 10mo

Role partition 1 : contient le système de fichiers, la racine

Partition 1 de type EXT4

Dossier plus volumineux : lib > modules, 57mo

- on peut diminuer sa taille en enlevant drivers pour ecran..

Programmes compilés dynamique avec uClibc

SysVinit utilisé pour démarrage services

TD7 – code sur arduino

Code dans boucle infinie

Les PD[0-7] sont définis dans avr/portpins.h inclu dans avr/io.h grâce à des #define

- Qu'est-ce que DDRD ? Que se passe t'il si je mets une led en D5 ? comment ajuster le programme correspondant ?

Ports D : digital pins

DDR : DATA DIRECTION REGISTER

DDRD : The Port D Data Direction Register

`DDRD = 0b11101111` //les bits ayant 0 seront input sinon output

Permet de dire si pin est input ou output

- Que se passe t'il si l'on ajoute dans le Makefile l'option '-Os' qui permet de mettre en œuvre les optimisations de gcc ?

Sans -Os : 975 octets

Avec -Os : 484 octets

Variable volatile a utiliser pour changer variables lors d'interruptions

`DDRB = 0b11111110` // met les broches # 1 à 7 du port B en sortie, la 0 en entrée

`DDRB |= 0b11111100` // configure les broches 2 à 7 en sortie sans modifier broches 0 et 1

Pendant interruptions les boutons sont plus déclencher -> faut utiliser interruption

`const uint8_t button _BV(PD4)` -> construit octet en fonction du port

`PORTD &= ~redLed` //on inverse redLed en gardant l'etat des autres ports

`if(PIND & button){` //button pressed

period >> 1 : divise par deux

```
cli();                  //disable global interrupts
sei();                  //enable global interrupts
```

Interruption

```
ISR(INT1_vect)          //see page 67 of datasheets
{
 EIMSK &= ~_BV(INT1);                    //disable INT1
 PORTD ^= greenLed;
 redIsBlinking = !redIsBlinking;

 _delay_ms(300);          //stabilization delay (unbounce)
 EIMSK |= _BV(INT1);                     //enable INT1
}
```

Avec PCINT

```
ISR(PCINT2_vect)          //see page 67 of datasheets
{
 PCMSK2 &= ~_BV(PCINT20);                  //disable PCINT2 from PD4 (PCINT20)

 if((PIND&button)){         //this is a raising edge
    isGreenLedOn = !isGreenLedOn;
    redIsBlinking = !redIsBlinking;
 }

 _delay_ms(300);            //stabilization delay (unbounce)
 PCMSK2 |= _BV(PCINT20);   //enable PCINT2 from PD4 (PCINT20)
}
```

PCINT peut réveiller en mode sleep et powerdown
INT : permet de choisir quel changement déclenche interruption

Interruption via timer

```
OCR1AH = 0x9D;
   OCR1AL = 0x09;              //time to count to
   TCCR1B |= 0x05;             // 1024 prescaler --> one 1 = 64us. 0x3D09 * 64us = 1s
   TCCR1B |= _BV(WGM12);         // select CTC (Clear Timer on Compare match) mode based on OCR1A (see table 16-4)
   OCR1A = 0x3D09; //1 seconde d'attente car 1024/16mhz = 64 us * 15625 = 1s = 1000000 = 0x3D09
   TIMSK1 |= _BV(OCIE1A); // enable timer interruptions

ISR(TIMER1_COMPA_vect) {
   //Desactive l'interuption
   TIMSK1 &= ~_BV(OCIE1A);
   PORTD ^= redLed;
   //On reactive
   TIMSK1 |= _BV(OCIE1A);
}
```

TD8 : FREE RTOS

On peut utiliser des tasks et définir leur priorité

```
xTaskHandle redBlink_handle;
   xTaskCreate                 //documented here: https://www.freertos.org/a00125.html
   (
      vRedBlinkLed,            //pointer function to the handler
      (const char*)"redBlink",   //naming the task
      configMINIMAL_STACK_SIZE,  //stack size
      NULL,                    //parameters of the handler
```

```
    1U,                 //priority
    &redBlink_handle    //address of task handler
  );
Puis vTaskStartScheduler();
```

Dans les tasks :

```
static void vGreenBlinkLed(void* pvParameters) {
  while (1)
    int xLastWakeTime = xTaskGetTickCount();
    if( xSemaphore != NULL ) {
      if (xSemaphoreTake(xSemaphore, portMAX_DELAY) == pdTRUE) {
        PORTD ^= greenLed; //PD3 on the micro controller is linked to D3 on the shield
        //vTaskDelay(0);  //passive Delay
      }else{
        PORTD &= ~greenLed; //turn off led
      }
    }else{
      PORTD &= ~greenLed; //turn off led
    }
    vTaskDelayUntil(&xLastWakeTime, 500 / portTICK_PERIOD_MS); //permet d'avoir vraie periodicite
  }
}
```

Taches sont périodiques, si temps d'execution infrieur au délais on peut considérer comme périodique

Sans délais on reste toujours dans la meme tache, donc celle avec plus grande priorité est toujours active, l'autre ne prend pas la main. Si pas de délais mais priorité égale les deux taches sont lancées

TD10 raspberry pi

Mount /dev/sdf1 /work/td10
cp -ar /work/td10/install/* /work/td06/rapberrypi3/overlay
cd /work/td06/buildroot.versionXX

En copiant nos fichiers dans overlay, en faisant make de buildroot il va construire image en prenant en compte nos fichiers dans overlay

➔ Donne sdcard.img

On peut dire systeme temps réel si on est presque sur a 100% qu'on aura les valeurs au temps voulu. Mais impossible a tester à 100%. On n'a pas la garantie que ce sera tout le temps le cas.
Seul moyen est d'avoir os écrit pour avoir temps réel et pas un linux même patché

TD11 raspberry

Grovepi agit comme framework, il fait abstraction des méthodes i2c pour les rendre plus simple à utiliser
digitalRead et digitalWrite pour agir avec capteurs et actionneurs
un potentiometre est une entrée analogique. La fonction utilisée est analogRead

probleme il marche pas sans convertisseur analogique numérique
raspy en a pas

le grovepi utilise ultrasonicRead
di_i2c utilise read_identified_i2c_block

Pourquoi n'est-t-il pas possible de faire fonctionner un code similaire directement en Python ou en C sur le micro-processeur de votre architecture ?
- Le Raspberry Pi ne possède pas de port analogique, du coup il ne pourrait pas lire ou pas correctement ce qui est envoyé par le capteur -> plus exactement, pas de convertisseur analogique numérique

Qu'est ce qui peut justifier de mettre en place une architecture mixte microprocesseur/microcontrôleur ?
Pour se baser sur le respect du temps on utilise microcontrôleur et avoir puissance de calcul microprocesseur
Pour utiliser le capteur qui calcule la distanc eil faut le micro controleur car il faut une mesure temporelle precise pour calculer la distance