Systèmes d'exploitation embarqués

Brique ROSE

Samuel Tardieu
sam@rfc1149.net

École Nationale Supérieure des Télécommunications

Système d'exploitation

Un système d'exploitation :

- fait le lien entre le logiciel (application) et le matériel
- abstrait certaines caractéristiques du matériel
- fournit des services communs (accès aux ressources, synchronisation, gestion de fichier)
- offre des possibilités de tests et de traces

Système embarqué

Généralement, un système embarqué :

- dispose de ressources limitées
- ne possède pas toujours de système de fichiers
- doit être le moins cher possible
- ne doit pas consommer d'énergie inutilement

Approche minimaliste

Dans une approche minimaliste, le système d'exploitation :

- initialise le matériel
- donne la main à l'application

Le système d'exploitation peut également être réduit à néant (carte nue).

Approche maximaliste

Dans une approche maximaliste, le système d'exploitation :

- fournit tous les services d'un ordinateur de bureau
- fournit des machines virtuelles
- fournit des bibliothèques graphiques

Exemple : le projet SPIF de l'ENST (Linux)

Approche intermédiaire

En général, le système d'exploitation fournit les services :

- de gestion de temps
- d'accès au réseau
- de gestion de la mémoire

Compromis

Des compromis doivent être faits :

- mono-programme / multi-programme
- mono-thread / multi-thread
- allocations statiques / allocation dynamique
- réservation des ressources / allocation des ressources à la demande

Exemple : PalmOS

- gestion de la mémoire simplifiée
- primitives de gestion de bases de données et de l'écran
- bibliothèques mathématiques
- applications minimalistes (philosophie du Palm)
- mono-application et mono-thread
- ne nécessite pas beaucoup de puissance (m68k 20MHz, ARM à 200MHz en émulation m68k)

Exemple : Windows CE

- tous les services d'un Windows
- fiabilité d'un Windows (mauvaise)
- facilité de portage des applications
- pas de gestion du temps-réel
- nécessite un processeur très puissant (ARM 400MHz)

Exemple : Symbian OS

- orienté *téléphonie*
- gestion des contacts
- gestion de réseaux divers (SMS, BlueTooth, GSM, TCP/IP)
- gestion multimedia
- synchronisation sur réseau lent *over the air*
- supporte Java (JavaPhone)
- nécessite moins de ressources que Windows CE

Exemple : Linux

- tous les services de Linux
- fiabilité de Linux (bonne)
- multi-applications
- très gourmand en ressources
- logiciel libre
- nécessite un processeur puissant (PowerPC 50MHz)

Exemple : RTEMS et eCos

- exécutif configurable pour ne garder que ce qui est nécessaire
- services de synchronisation
- gestion du temps
- entrées/sorties
- gestion du réseau
- logiciel libre
- ne nécessite pas beaucoup de puissance (m68k à 33MHz)

Exemple : Forth

- langage de programmation et système à lui tout seul
- mono-application mais multi-threads en mode coopératif
- permet le test interactif
- ne nécessite pas beaucoup de ressources (PIC 16f876, 2k de programme, 80 octets de RAM)

Caractéristiques

Avant de choisir son système, il faut définir :

- le prix de revient maximum de chaque unité
- le temps de réponse aux actions de l'utilisateur
- la capacité totale de traitement
- la disponibilité et la possibilité de le modifier
- le support disponible et la fiabilité du fournisseur

Rappels de compilation croisée

Définitions :

- L'**hôte** est la machine sur laquelle on compile
- La **cible** est la machine pour laquelle on compile

Différences possibles :

- Type de microprocesseur (*big/little endian*)
- Taille des mots (représentation des données de la cible)
- Système d'exploitation (conventions d'appel)

Moniteur

Le moniteur :

- configure la cible (mémoire, ports de communication)
- permet de dialoguer et de recevoir des commandes
- autorise le diagnostic
- permet de charger un autre programme
- permet de reprogrammer la cible (flash)

Exemples de moniteurs

- OpenFirmware : Sun Sparc et Apple Macintosh récents
- TinyBoot : version réduite d'OpenFirmware
- PPCboot : plate-forme PowerPC embarquée
- Différents *bootloaders* : permettent uniquement de charger du code
- Pour la programmation ombilicale, petits moniteurs *maison*

OpenFirmware

- Standard IEEE 1275
- Interpréteur de *bytecode* basé sur Forth (fcode tokens)
- Indépendant de l'architecture (Sparc et Macintosh)
- Mots standards pour examiner le bus
- Du *bytecode* est chargé depuis les cartes d'extension pour enrichir le système
- En développement pour PC (OpenBios)

Exemple OpenFirmware

```
Type 'go' to resume
ok devalias
screen      /pci@1f,0/pci@1,1/SUNW,m64B@2
net         /pci@1f,0/pci@1,1/network@1,1
[...]
ok dev screen
ok dimensions .s
1280 1024
ok dev net
ok watch-net
Internal loopback test --
```

Transfert de contrôle

Le moniteur peut également

- charger un autre programme (noyau)
- passer des structures de données et des méthodes au noyau (factorisation des fonctionalités)
- reprendre la main si le noyau quitte
- reprendre la main sur interruption (debug)
- reprendre la main en cas de blocage de la machine

Méthodes de communication

Un moniteur utilise

- un moyen de communication standard
  - liaison série, clavier
  - cela monopolise un périphérique qu'on ne peut plus tester
- ou un moyen de communication *ad-hoc*
  - protocole *1 fil*
  - cela libère les autres périphériques

Configuration d'un port

La configuration d'un système de communication dans un moniteur est différente d'avec un OS :

- les spécificités matérielles ne sont pas masquées
- la manipulation se fait port par port et parfois bit par bit

On échappe rarement à la lecture de la *datasheet* du système utilisé.

Exemple : RS232sur PIC 16f876

Les différentes étapes sont :

- sélectionner la vitesse du port $B = \frac{F_{osc}}{N(X+1)}$ $(X \in [0, 255])$ et le mode $N = 16$ ou $N = 64$
- activer le mode asynchrone
- sélectionner le format (8 ou 9 bits)
- configurer les interruptions pour l'émission
- configurer les interruptions pour la réception
- configurer les interruptions pour les périphériques
- configurer les interruptions globales

Protocole *1 fil*

- En réalité, 1 fil plus la masse
- Protocole *half-duplex*
- Utilisation de *polling* : le maître demande régulièrement à l'esclave s'il a quelque chose à transmettre
- Nécessité de synchroniser (statiquement ou dynamiquement) les horloges

Exemple de protocole *1 fil*

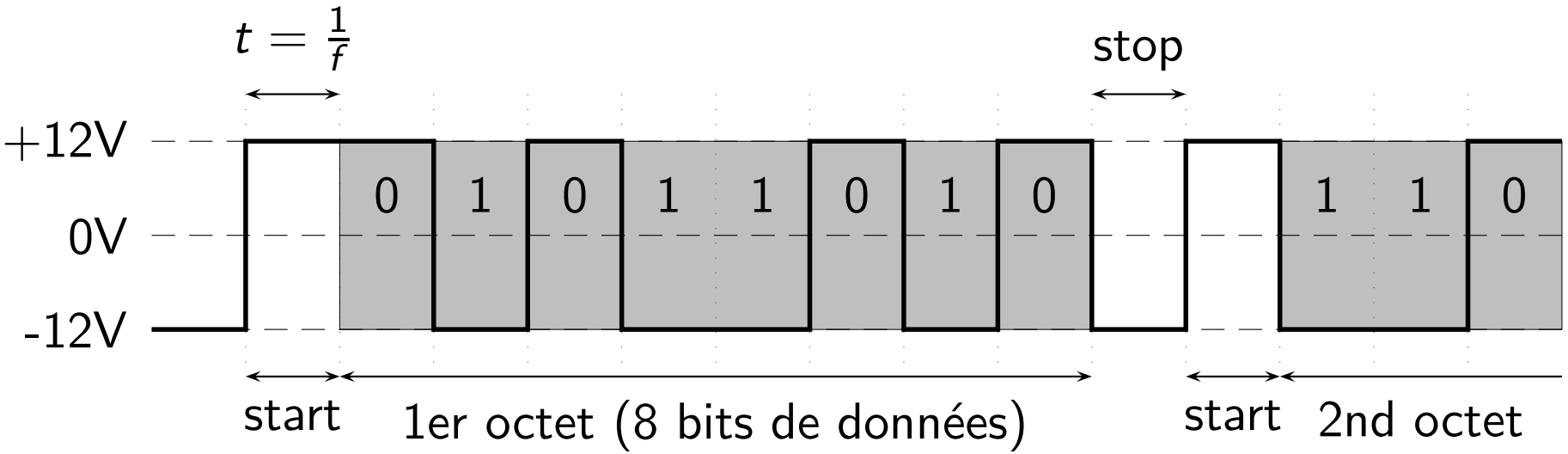
- Le maître envoie un caractère régulièrement, ou **?** pour rien
- Le maître envoie un **:** suivi d'un **:** ou **?** pour transmettre ces deux derniers (échappement)
- Après chaque caractère reçu, l'esclave envoie un caractère ou **?** s'il n'a rien à envoyer
- L'esclave utilise les mêmes séquences d'échappement

Protocole RS232

- Protocole *full-duplex* (TX pour la transmission et RX pour la réception)
- L'état au repos de la ligne est -12V
- Un **bit de start** (ligne à +12V) permet de détecter le début de la transmission
- Chaque bit est échantillonné plusieurs fois (3 ou 5) pour vérifier la bonne synchronisation des horloges
- Un **bit de stop** (ligne à -12V) permet de vérifier la cohérence des vitesses (avant le prochain bit de start – *framing*)

Exemple de transmission RS232

- Transmission du caractère “Z” de code ASCII 90, soit 5a en hexadécimal ou 01011010 en binaire
- Transmission à f bits par secondes (par exemple, $f = 9600$bps)
- 1 bit de start, 1 bit de stop (parfois 1,5 ou 2), pas de parité (parfois parité paire ou impaire)
- Vitesse effective : $\frac{f}{10}$ octets par seconde (surcoût de 25%)



Programmation des composants

Certains composants sont programmables (une ou plusieurs fois) :

- en utilisant un programmateur
- ou en utilisant la programmation *in-situ*
- avec ou sans tension d'alimentation spéciale
- parfois depuis du code qu'ils exécutent eux-mêmes

La dernière classe est la plus intéressante.

Avantage de la reprogrammation

- Il est possible d'installer un moniteur sur le composant
- Aucun appareillage n'est nécessaire pour le reprogrammer : il suffit de disposer d'un moniteur
- Il est possible de tester facilement des changements
- Il est possible de mettre à jour des appareils en production s'ils ont été prévus pour cela

Moniteur complet minimaliste

Un moniteur complet minimaliste pourrait

- accepter un entier et le placer sur une pile
- écrire le deuxième entier à partir du sommet à l'adresse se trouvant sur le sommet
- sauter à l'adresse se trouvant sur le sommet

Un tel moniteur peut

- prendre le contrôle au boot
- transférer le contrôle à un programme si une condition est vraie (bouclage I/O)

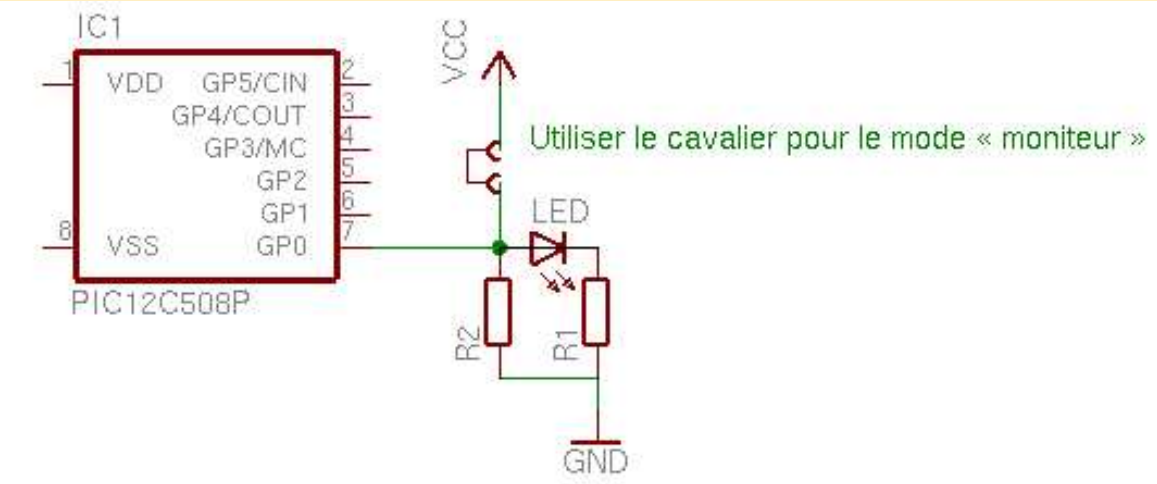
Contrôle du moniteur

Problème

Comment faire lorsqu'on ne dispose pas d'entrée disponible à dédier au moniteur ?

Solution

On peut utiliser une sortie comme entrée en la testant au moment du démarrage.



On choisit bien entendu $R2 >> R1$.

Chargement d'un noyau

Un noyau peut être chargé :

- depuis un support de masse (disque dur, ROM, EEPROM, flash)
- depuis une laison avec un système extérieur (série, ethernet, WiFi)

Il peut être compressé ou décompressé, chiffré, signé, etc.
Le moniteur doit savoir accéder à ces supports.

Anatomie d'une application

Une application (par exemple un noyau) est composée de plusieurs sections : (exemple de ELF)

- **.text** : le code exécutable ; il sera logé à un endroit égal ou différent de celui où il se trouve stocké, en mémoire vive ou morte
- **.data** : les données initialisées ; elles seront copiées en mémoire vive avant de lancer le sous-programme principal
- **.bss** : les données non initialisées ; une zone de mémoire vive de la bonne taille sera initialisée à zéro

Exemple : PC sous FreeBSD

Le boot de FreeBSD se fait en différentes étapes :

- le moniteur du PC (BIOS) charge le premier secteur (`boot0`) du périphérique de démarrage et l'exécute
- `boot0` charge le premier secteur de la partition de démarrage choisie et l'exécute (`boot1`)
- `boot1` extrait `boot2` de la partition de démarrage (à un endroit fixe) et l'exécute
- `boot2` charge et exécute `/boot/loader` (moniteur contenant un interpréteur de commande Forth) ou directement un noyau

Chargement à travers le réseau

Le chargement par dessus TCP/IP implique :

- l'obtention d'une adresse IP
- la communication avec un serveur de fichiers
- la récupération d'un noyau adéquat

Différents protocoles interviennent.

Obtention d'une adresse IP

L'adresse IP :

- peut être obtenue par bootp, DHCP, bootparam
- est accompagnée de paramètres supplémentaires (masque de sous-réseau, passerelle, serveurs de nom, domaine courant)
- peut être accompagnée d'informations de boot (serveur, protocole, paramètres)

Obtention d'un noyau

Le noyau peut être obtenu par

- **TFTP** : protocole trivial basé sur UDP, peu sécurisé
- **NFS** : montage d'un système de fichiers sur lequel se trouve le noyau
- **FTP, samba** : tout système peut être *a priori* utilisé, ce n'est qu'une convention entre le système embarqué et le serveur

Compression de l'application

L'application peut être accessible par le moniteur à un format compressé :

- le moniteur doit savoir la décompresser
- l'encombrement statique du moniteur est augmenté
- la mémoire nécessaire pour la décompression doit être prise en compte :
  - environ 32Ko pour le format gzip
  - environ 900Ko pour le format bzip2

Systèmes sans stockage

Les systèmes sans stockage sont

- soit autonomes
- soit dépendants de ressources extérieures

Sur un système Unix, les accès à certaines partitions (voire toutes) et le swap peuvent avoir lieu par dessus NFS.

Stockage en mémoire flash

La mémoire flash connaît les mêmes contraintes (en plus prononcées) qu'un disque dur :

- l'écriture se fait par bloc (secteur sur un disque) entier
- plus on écrit au même endroit, plus il s'use

Par contre :

- la mémoire flash coûte beaucoup plus cher et est beaucoup plus fragile
- la mémoire flash consomme beaucoup moins de courant et nécessite moins de place

Gestion de la mémoire flash

Propriétés souhaitables d'un système de gestion de fichiers en mémoire flash :

- répartition homogène des écritures (*wear levelling*)
- cohérence du système de fichier, même en cas de coupure de courant en plein milieu d'une opération
- souplesse d'un système de fichiers traditionnel

Solutions disponibles

- systèmes de fichiers basiques (pas de *wear levelling*, pas sûrs)
- standards PCMCIA FTL (*Flash Translation Layer*) et NFTL (brevetés, donc inutilisables)
- système de fichiers JFFS (*Journalling Flash File System*) et JFFS2 (Linux, eCos en cours)

JFFS/JFSS2

- les données compressées sont écrites séquentiellement dans la mémoire flash (*journalling*)
- la mémoire flash est entièrement analysée au moment du montage
- seules les méta-données qui ne peuvent être reconstituées rapidement sont gardées en mémoire
- pour pouvoir libérer la totalité d'un bloc, les informations non obsolètes sont déplacées
- il est nécessaire de garder quelques blocs libres en permanence pour cette gestion

Limitations de JFFS/JFFS2

- Les programmes sont copiés en RAM avant d'être exécutés (pas de XIP *eXecute In Place*) ; XIP et compression sont des facilités incompatibles
- L'appel système `mmap()` passe par un intermédiaire en RAM pour la même raison
- La détection et gestion des erreurs est primitive (pas de correction, uniquement une signalisation des mauvais blocs)

Intérêt de JFFS sur iPaq

- Sous PocketPC (Windows CE), la mémoire flash (32Mo) sert de stockage pour le système d'exploitation. La RAM est partagée entre le stockage des données (et applications) de l'utilisateur et la mémoire nécessaire pour l'exécution des programmes.
- Sous Linux (distribution Familiar), la mémoire flash sert de stockage du système d'exploitation et des données utilisateurs ; ces données survivent à une perte de courant, contrairement aux informations stockées uniquement en RAM.

Watchdog

- But : détecter une erreur logique dans le programme
- Principe : si on ne se manifeste pas pendant un certain temps auprès du watchdog, il réinitialise le processeur
- Contraintes :
  - il faut être sûr que le programme avance
  - il faut se manifester le moins possible
  - cela oblige à réfléchir au *timing* des différentes parties du programme

Watchdog logiciel et matériel

Le meilleur watchdog est matériel :

- il se déclenche toujours
- il ne peut pas être désactivé par le logiciel
- mais il n'est pas souple (pas de supervision)

Un watchdog logiciel peut être utile :

- détection de surcharge d'un noyau multi-tâches
- gestion de supervision

Gestion de l'énergie

Un système embarqué doit gérer l'énergie

- pour être économique :
  - moins de consommation
  - alimentation avec un courant plus faible
- pour être citoyen :
  - moins de consommation de ressources non renouvelables
  - utilisation de piles plus propres
  - recyclage des déchets moins fréquent

Mode *veille*

La plupart des composants disposent d'un mode *veille* :

- les fonctions vitales sont endormies
- certains événements (interruptions, timer) peuvent les réveiller
- parfois, une deuxième horloge est nécessaire pour assurer la veille du composant

Mais la veille se fait avant tout en évitant les *gadgets* : leds, affichage de l'heure, etc.

Variété des consommations

La carte SPIF, équipant les robots du département INFRES de l'ENST :

- consomme 4W à vide au repos
- consomme 7W à pleine puissance
- consomme 15W à pleine puissance en bloquant les moteurs