

Linux embarqué – INSA Toulouse

Fabien Lahoudere (fabien.lahoudere@openwide.fr)

- Décrire les éléments d'une distribution Linux minimale à base de kernel 3.4 et Busybox
- Construire une distribution à l'aide d'OpenEmbedded
- La carte utilisée est basée sur un module CPUIMX51SD de Eukrea
 - Processeur i.MX515 de Freescale (Cortex A8)
 - 256 Mo SDRAM, 1 Go flash NAND
 - 2 ports RS-232, Ethernet, USB OTG
 - GPIO, SPI, I2C, ...
 - JTAG
 - Bootloader Barebox
- Développement d'un driver Linux (pilotage servo)

Rappels sur GNU/Linux

Origines, licences, architecture

- Linux est une implémentation libre d'UNIX diffusé sous licence GPL
- Inspiré des deux versions: AT&T et BSD
- Les principes d'UNIX sont respectés:
 - Simplicité, modularité, respect des standards, ouverture
 - Deux espaces de mémoire: noyau / utilisateur
 - Le noyau permet d'accéder au matériel (pilotes, appels système)
 - Tout composant est un fichier: répertoire, périphérique, élément de communication, etc. (organisation arborescente)
 - Puissance de la «ligne de commande» (*shell* et *regexpr*)

- Noyau monolithique (en un seul fichier) + modules dynamiques
- Création des processus via `fork()` et `exec()`
- Multi-threading
- Nombreuses piles réseau (IPv4, IPv6, Ethernet, etc.)
- Organisation des fichiers arborescente à partir de la racine (`/`), montage et démontage *logique* (`mount`)
- Notion de super-utilisateur (*root*), groupes, et utilisateurs

- Linux est fortement lié au projet GNU (GNU is Not UNIX) de Richard Stallman (MIT, années 80)
- Le nom officiel de l'OS Linux est *GNU/Linux* car « Linux » correspond uniquement à la partie noyau
- Libre de toute licence source par rapport à AT&T
- Internet a également fortement contribué au succès de Linux
- Pour un non initié, la personnalité de Linus Torvalds est plus « rassurante » que celle de R. Stallman :-)

Les *parents* de Linux !



- GPL = General Public License
- On la surnomme également « copyleft »
- La GPL v2 (1991) est la plus répandue (ex: noyau Linux)
- La licence s'applique uniquement en cas de redistribution
- Un code source utilisant du code GPL est du travail dérivé et doit être publié
- Publication: celui qui reçoit la version binaire peut obtenir le code source
- Pas de lien (ld) possible entre du code GPL et du code « propriétaire » !

- La GPL est complexe à gérer dans un contexte industriel → création de la LGPL
- Le lien avec du code propriétaire est possible avec la LGPL (*Lesser/Library* GPL) !
- En majeure partie, les bibliothèques système sont diffusées sous LGPL (exemple: GNU-libc)
- Dans le cas d'une application propriétaire il faut donc vérifier qu'aucune bibliothèque « liée » n'est GPL
- Le lien dynamique n'affranchit pas de la licence sauf dans des cas très particuliers

- Nouvelle version sortie en 2007
- Oblige à fournir les éléments pour construire un logiciel fonctionnel => réponse à la « Tivoisation »
- La GPL v2 demande uniquement la publication des sources à celui qui a reçu le binaire
- La GPL v3 ne sera pas utilisée pour le noyau Linux.
- Voir:
<http://www.gnu.org/licenses/quick-guide-gplv3.fr.html>

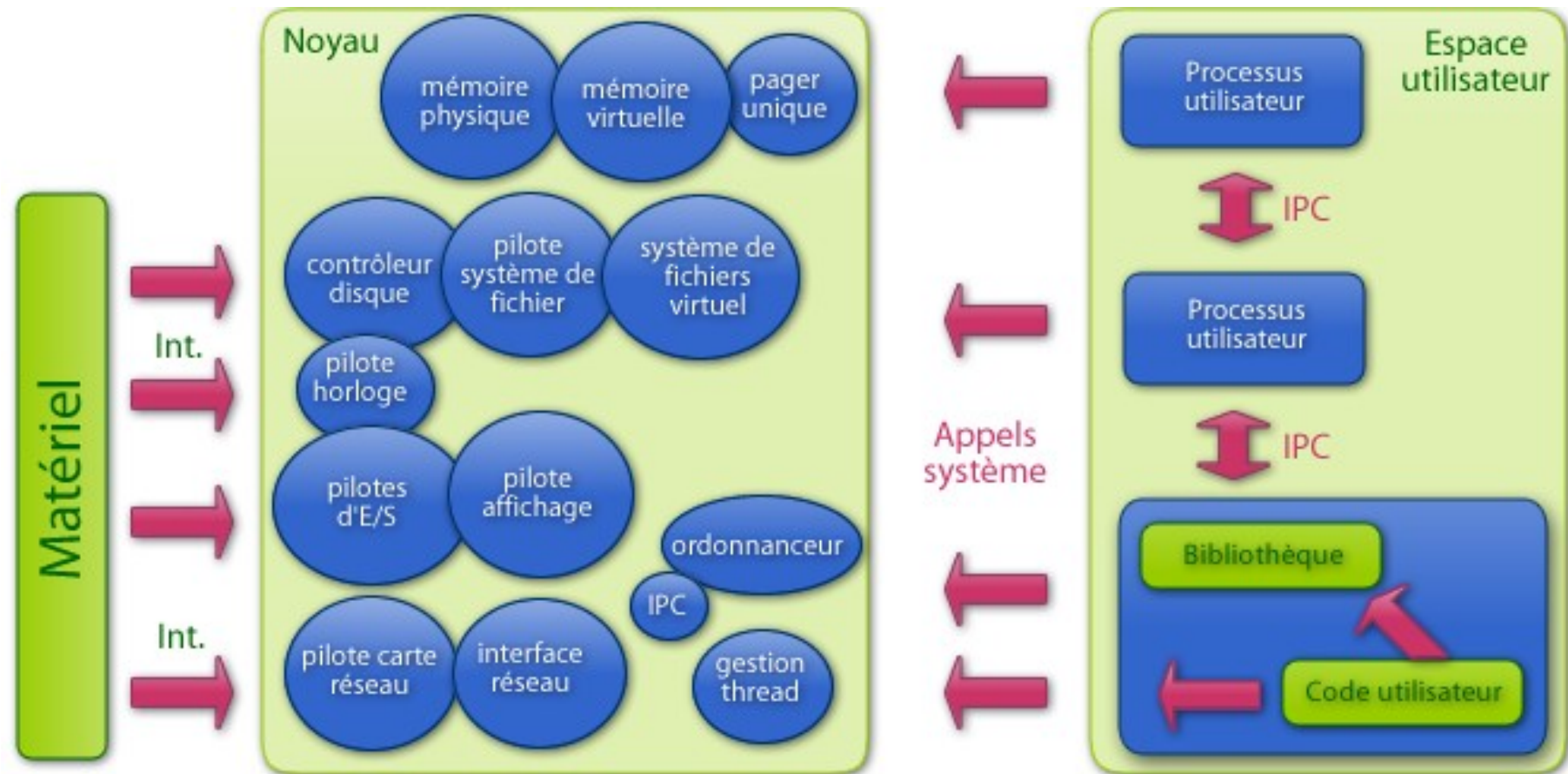
- Dans l'espace noyau (pilotes), SEULE la GPL s'applique (en théorie) !

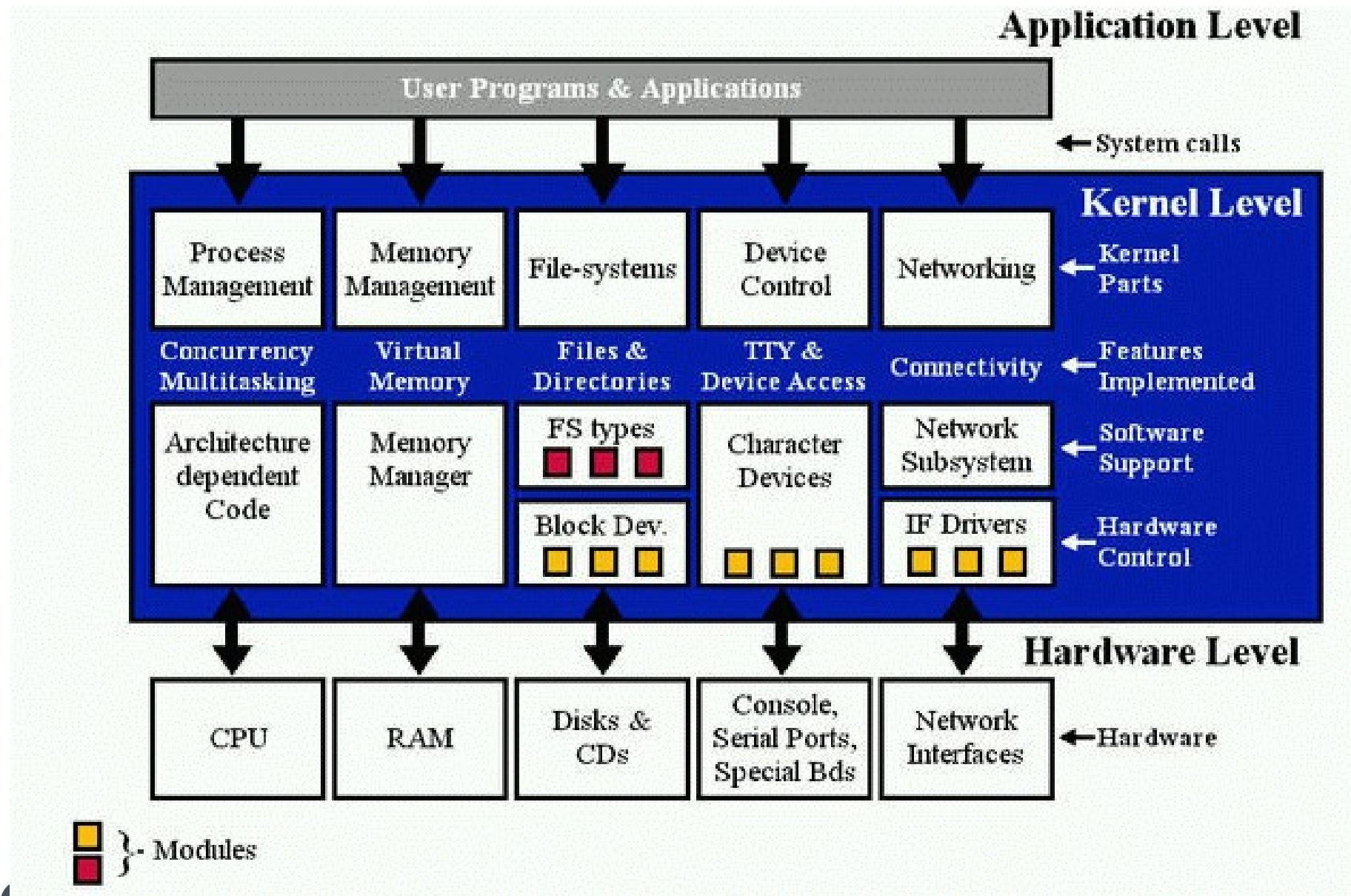
You cannot use kernel headers file to create non GPL'd binaries (Linus Torvalds)

- Certaines fonctions non disponibles si la licence n'est pas GPL
- En pratique: tolérance si le pilote n'a pas été créé pour Linux (cas du portage) => nVidia, Broadcom, ...
- Cependant les pilotes binaires posent des soucis techniques vu qu'un pilote fonctionne pour la version de noyau utilisée pour la compilation

Présentation de l'architecture Linux

Architecture du système





- Les éléments fondamentaux :
 - Le noyau: en théorie, interface unique avec le matériel, API de programmation spécifique (modules Linux)
 - Le «root filesystem» (root-fs): les commandes et fichiers système, communs à (presque) toutes les versions d'UNIX (API standard => POSIX)
- Un système Linux est obligatoirement l'association noyau + root-fs
- Embarquer: optimiser le noyau + construire un root-fs léger

- En 4 étapes
 - Chargement du bootloader qui charge le noyau Linux
 - Le noyau Linux statique initialise les périphériques
 - Le noyau monte le root-fs et exécute `/sbin/init`
 - Le processus init (PID=1) lit `/etc/inittab` et exécute le script (shell) de démarrage des services «user-space» (rc= run command)

- Sur x86, le BIOS démarre un bootloader comme GRUB ou LILO
- Pour les autres architectures, le bootloader fait office de BIOS
 - U-Boot ou Barebox
 - Red Boot
 - PPC Boot
 - CFE
- Le bootloader démarre le noyau Linux (partie statique)

- Chargement de la partie statique du noyau:
 - vmlinux
 - zImage/bzImage
 - uImage (U-Boot)
- Initialisation du CPU et services fondamentaux (ordonnanceur, gestion mémoire, disque, flash, ...)
- Montage du root-fs sur « / »
- Exécution du premier processus (init, PID=1) correspondant au fichier /sbin/init

- Le «père» des processus du système
- Lit le fichier `/etc/inittab` au démarrage:
 `id:5:initdefault:`
 `si::sysinit:/etc/rc.d/rc.sysinit`
 ...
 - `id`: niveau de démarrage (5= X11)
 - `si`: chemin du script de démarrage (`rc.sysinit`)
 - Les noms (`rc*`) peuvent varier suivant les distributions

- Organisation commune à 90% entre les UNIX
- Quelques spécificités GNU/Linux et distribution
- Les fichiers communs:
 - `/bin`: binaires communs
 - `/lib`: bibliothèques et modules noyau
 - `/sbin`: binaires « système »
 - `/etc`: fichiers de configuration
 - `/dev`: nœuds d'accès aux périphériques (nodes)
 - `/var`: fichiers *variables*: `log`, `spool`, `mail`, ...
 - `/usr`: reproduit / pour `bin`, `lib`, `sbin`, etc

- /opt: pour les programmes externes (ex: OpenOffice)
- /home: accueille les répertoires des utilisateurs (home-directory)
- Quelques répertoires *spéciaux*
 - /root: home-directory de l'utilisateur *root* (*pas dans OE*)
 - /media: point de montage des volumes amovibles
 - /proc: système de fichier virtuel (état du système)
 - /sys: idem pour les périphériques connectés (2.6)
 - /boot: noyau statique (vmlinux, ulmage, ...)

- Modules noyau: `.ko` (Kernel Object) dans `/lib/modules`
- Un sous-répertoire par version de noyau, exemple: `2.6.30.2-test1` ← Local version
- Les modules binaires sont liés à la version du noyau !
- Dans le répertoire du noyau:
 - `kernel`: arborescence des modules standards
 - `extra`: modules ajoutés (pilotes externes aux sources)
 - `modules.dep`: fichier des dépendances créé par `depmod -a`

- Système de fichier *virtuel* géré par le noyau (origine SVR4)
- Ce n'est pas un véritable système de fichiers: ni sur le disque ni en mémoire (pas d'effacement possible)
- En lecture/écriture
- Intérêt: manipuler des variables système comme des fichiers avec les commandes classiques :
 - cat, echo, grep
- Exemples:
 - /proc/version: version du noyau
 - /proc/cpuinfo: type(s) de processeur(s)

- /proc/interrupts: interruptions
- /proc/<pid>: répertoire décrivant le processus associé au pid
- /proc/mounts: partitions montées
- /proc/sys/net/ipv4/ip_forward: forwarding IP
 - # echo 1 > /proc/sys/net/ipv4/ip_forward
- /proc/modules: liste des modules noyau chargés
- Nombreuses commandes basées sur /proc :
 - lsmod, lspci, ps, top, ...
- Il est aisé de créer une entrée /proc dans un nouveau pilote

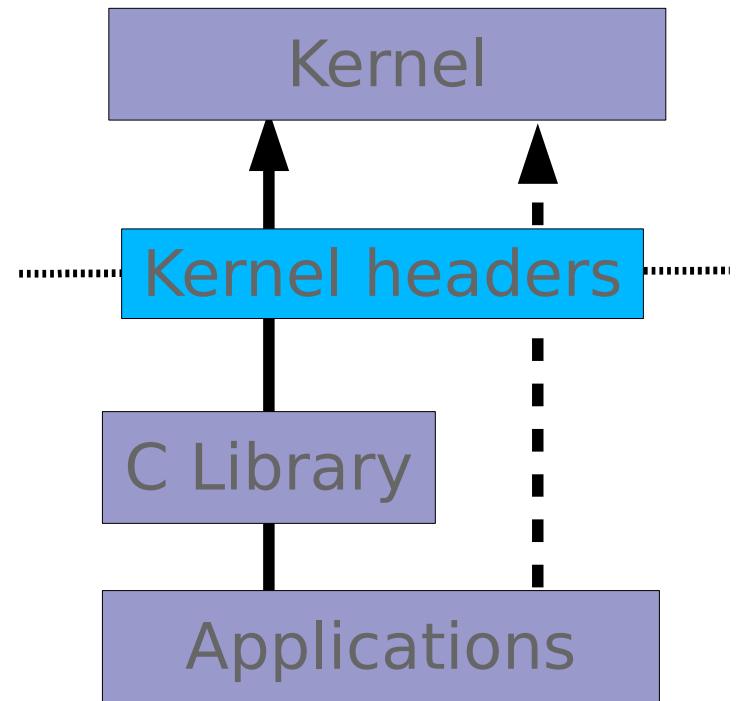
- Introduit dans le noyau 2.6 (2003) => sysfs
- Vue synthétique des périphériques connectés
 - /sys/class (utilisé par UDEV)
 - /sys/modules
 - /sys/bus
- But: mieux gérer l'ajout/suppression dynamique des périphériques (hotplug)
- Utilisé par UDEV pour créer dynamiquement les entrées dans /dev
- Quelques recouvrements avec /proc (bus PCI, USB, ...)

Création d'une distribution embarquée

- Chaîne de compilation croisée :
 - Gcc
 - as
 - ld
 - LibC
 - ...
- Bootloader (U-Boot)
- Noyau Linux adapté
- Commandes Linux (sh, ls, cp, etc.)
- Outil de génération (BR, OE, ...)

- Un point *très* complexe !
- Nécessité de construire une chaîne croisée :
 - GCC
 - Binutils (as, ld, ...)
 - Dépendances avec le noyau (*system calls*, ...) => erreur « Kernel too old »
 - Choix d'une libC => Glibc, uClibc, Eglibc, ...
 - GDB
 - Toute autre bibliothèque utilisée => libstdc++
 - Dépendances avec le compilateur hôte

- Interaction entre la libC et le noyau Linux
 - Appels systèmes (nombre, définition)
 - Constantes
 - Structures de données, etc.
- Compiler la libC – et certaines applications - nécessite les en-tête du noyau
- Disponibles dans `<linux/...>` et `<asm/...>` et d'autres répertoires des sources du noyau (`include, ...`)



- Utiliser un compilateur binaire :
 - ELDK: <http://www.denx.de/wiki/DULG/ELDK>
 - Code Sourcery :
<https://sourcery.mentor.com/sgpp/lite/arm/portal/release1803>
 - Installation simple
 - Support (payant) possible
 - Configuration connue => support par les forums
- Par contre:
 - Versions des composants figées
 - Non utilisation des possibilités du CPU
 - Choix libC limité

- Construire un compilateur
 - Crosstool => obsolète
 - Crosstool-NG => assez complexe à prendre en main
 - Buildroot / OpenEmbedded
- Aucun n'est « plug and play »
- La mise au point peut prendre des jours, voire plus !
- Binaires produits : `arm-linux-*` (gcc, as, ld, ar, nm, ...)

- Universal Bootloader
- Développement géré par DENX Software
<http://www.denx.de/wiki/DULG/Manual>
- Support d'un grand nombre de cartes ARM, PowerPC, MIPS, SH4, ...
- Basé sur un principe de variables d'environnement (setenv, printenv, saveenv, ...)
- Possibilité/nécessité d'écrire des macros → chargées depuis le PC hôte
- Support d'adresses statiques ou DHCP, protocole TFTP
- Support des flash NOR, NAND
- Nouvelle version U-Boot v2 → Barebox

- Extraction du noyau 3.6

```
$ tar xzf dl/linux-3.6.1.tar.gz
```

```
$ cd linux-3.6.1
```

- Configuration

```
$ make menuconfig
```

Modification de la config

- Compilation

Nécessite mkimage (produit par U-Boot)

```
$ make uImage V=1
```

Mode « verbeux »

- Le noyau produit correspond aux fichiers :

- arch/arm/boot/zImage ou bien arch/arm/boot/uImage (pour U-Boot)

- GNU/Linux basé sur « coreutils » est en général trop volumineux pour l'embarqué

```
$ ls -l /bin/bash
```

```
-rwxr-xr-x 1 root root 877480 21 mai 2010 /bin/bash
```

- Busybox remplace la majorité des commandes Linux par des versions « réduites »

```
$ ls -l /bin/busybox
```

```
-rwsr-xr-x 1 root root 670856 15 mars 09:45 /bin/busybox
```

- 95 % des distributions « Linux embarqué » utilisent Busybox
- Simple, léger, portable
- Diffusé sous licence GPLv2

- Busybox utilise la même procédure de compilation croisée que le noyau (variables ARCH et CROSS_COMPILE)
- Configuration de Busybox (par défaut)

```
$ tar xjf dl/busybox-1.17.4.tar.bz2
```

```
$ cd busybox-1.17.4
```

```
$ make defconfig
```

```
$ make menuconfig (ajout de l'option  
-march=armv4t)
```

- Compilation

```
$ make
```

- Installation

```
$ mkdir $HOME/rootfs_sodimm
```

```
$ make CONFIG_PREFIX=$HOME/rootfs_sodimm  
install
```

- Le seul exécutable est `/bin/busybox` et utilise `libc.so` et `libm.so` (voir avec `arm-linux-ldd` si il existe)
- Plusieurs solutions
 - Copie des bibliothèques « à la main »
 - Copie automatique des bibliothèques de la chaîne croisée (BR / OE)
 - Utilisation du script `mklibs` (pas toujours fiable)

```
$ cd rootfs_sodimm
```

```
$ mkdir lib
```

```
$ mklibs --target arm-none-linux-gnueabi -D -L $HOME/arm-  
2010.09/arm-none-linux-gnueabi/libc/armv4t/lib -d lib  
bin/busybox
```

- Création du fichier de démarrage : `etc/init.d/rcS`

```
#!/bin/sh
```

```
mount -a      # montage des filesystems décrits dans fstab
```

```
mdev -s      # remplissage de /dev
```

- Création de la liste des fs montés : `etc/fstab`

```
proc    /proc                proc    defaults          0          0
```

```
sysfs   /sys                 sysfs   defaults           0          0
```

- On rend rcS exécutable

```
$ chmod +x etc/init.d/rcS
```

- Création des répertoires `/dev`, `/proc`, `/sys`

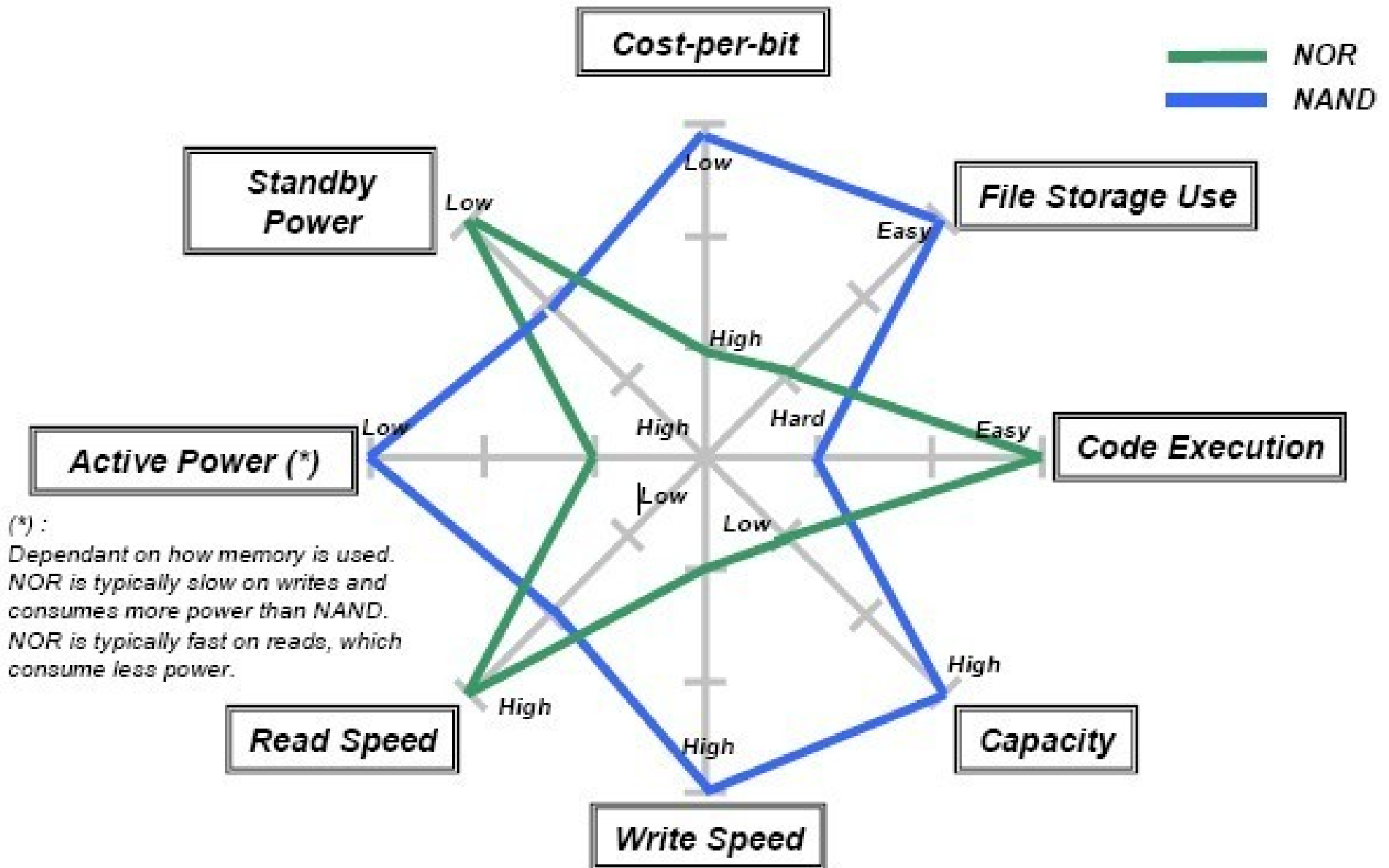
```
$ cd rootfs_sodimm; mkdir dev proc sys
```

```
S sudo mknod dev/console c 5 1
```

Utilisation de la mémoire flash

- Technologie dérivée de l'EEPROM, mais plus rapide à programmer, d'où le nom !
- Cellule de base = transistor MOS (Metal Oxyde Semiconductor)
- Deux catégories :
 - NOR, inventée par un ingénieur de Toshiba , commercialisée par Intel en 1988 => accès aléatoire (direct)
 - NAND, commercialisée par Toshiba en 1989 => accès séquentiel, moins fiable => ECC (Error Correcting Code)
- La NOR est normalisée par un jeu de commandes *standardisé* (CFI = Common Flash Interface)

- La NOR est plus onéreuse que la NAND mais mieux adaptée à l'exécution de code
- La NAND est de plus en plus fréquente à cause du coût
- La mémoire flash est caractérisée par deux paramètres (outre sa capacité)
 - Taille de page (page size)
 - Taille de bloc (block size ou erase size) = plusieurs pages
- Ces paramètres font partie de la configuration du système (à préciser dans Buildroot => *Target filesystem options*)



- Chargement par TFTP à une adresse RAM (0x200000)
- Copie sur la flash (0x34080000, APRES U-Boot !)
- Écriture des secteurs 2 à 32:

```
# protect off 1:2-32
```

```
# erase 1:2-32
```

```
# cp.b 200000 34080000  
taille_transfert
```

```
# protect on 1:2-32
```

- Test de démarrage :
bootm 34080000

- Utilisation de la commande nand
 - `help nand`
 - `nand info`
 - `nand read`
 - `nand write`
 - `nand write.jffs2`
 - ...
- Pas de protection/dé-protection
- Utilisation de « macros » U-boot recommandée !

- On *flashe* les fichiers uImage et rootfs.jffs2
- Sur la cible, on utilise les commandes + macros suivantes :

```
# tftp [${loadaddr} uImage]
```

```
# run kern_erase
```

```
# run kern_write
```

```
# tftp ${loadaddr} rootfs.jffs2
```

```
# run root_erase
```

```
# run root_write
```

- Les premiers paramètres de bootargs :
`# setenv bootargs console=ttyACM0,115200 mem=64M`
- Options pour la NAND :
`root=/dev/mtdblock2 rootfstype=jffs2`
- Démarrage :
`# run nandboot`

- Utilisation du pilote MTD (Memory Technology Device)
- Accès par le menu *Device drivers*
- Configuration NOR/NAND différente
 - La NOR est vue comme *physical map* (adresse de base et taille)
 - La NOR est normalisée par CFI (Common Flash Interface)
 - La NAND nécessite un pilote particulier dépendant de l'architecture
- Découpage de la flash statique (dans le noyau) ou bien dynamique via `mtdparts`

- Paramètre mtdparts à ajouter à bootargs
mtdparts=physmap-flash.0:0x80000(u-boot)ro,0x7c0000(kernel),-(rootfs)
physmap-flash.0 est l'identifiant de la flash

```
.config - Linux Kernel v2.6.30 Configuration

Device Drivers

Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

^(-)
< > Connector - unified userspace <-> kernelspace linker --->
[*] Memory Technology Device (MTD) support --->
< > Parallel port support --->
[*] Block devices --->
[*] Misc devices --->
< > ATA/ATAPI/MFM/RLL support --->
SCSI device support --->
< > Serial ATA (prod) and Parallel ATA (experimental) drivers --
[ ] Multiple devices driver support (RAID and LVM) --->
[*] Network device support --->

v(+)

<Select>  < Exit >  < Help >
```



```
.config - Linux Kernel v2.6.30 Configuration

Memory Technology Device (MTD) support
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

-- Memory Technology Device (MTD) support
[ ]   Debugging
< >   MTD concatenating support
[*]   MTD partitioning support
< >   MTD tests support
< >   RedBoot partition table parsing
[*]   Command line partition table parsing
< >   ARM Firmware Suite partition parsing
< >   TI AR7 partitioning support
*** User Modules And Translation Layers ***
<*>   Direct char device access to MTD devices
-*-*   Common interface to block layer for MTD 'translation layers
<*>   Caching block device access to MTD devices
< >   FTL (Flash Translation Layer) support
v(+)
```

<Select> < Exit > < Help >

```
.config - Linux Kernel v2.6.30 Configuration

RAM/ROM/Flash chip drivers

Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

<*> Detect flash chips by Common Flash Interface (CFI) probe
< > Detect non-CFI AMD/JEDEC-compatible flash chips
[*] Flash chip driver advanced configuration options
    Flash cmd/query data swapping (NO) --->
[ ] Specific CFI Flash geometry selection
[ ] Protection Registers aka one-time programmable (OTP) bits
<*> Support for Intel/Sharp flash chips
< > Support for AMD/Fujitsu/Spansion flash chips
< > Support for ST (Advanced Architecture) flash chips
< > Support for RAM chips in bus mapping
v(+)
```

<Select> < Exit > < Help >

```
.config - Linux Kernel v2.6.30 Configuration

Mapping drivers for chip access

Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

[ ] Support non-linear mappings of flash chips
<*> Flash device in physical memory map
[*] Physmap compat support
(0x34000000) Physical start address of flash mapping
(0x40000000) Physical length of flash mapping
(2) Bank width in octets (NEW)
< > CFI Flash device mapped on ARM Integrator/P720T
< > Map driver for platform device RAM (mtd-ram)

<Select>  < Exit >  < Help >
```

```
.config - Linux Kernel v2.6.28-pragmatec Configuration

Memory Technology Device (MTD) support
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

^(-)
<*>  INFTL (Inverse NAND Flash Translation Layer) support
< >  Resident Flash Disk (Flash Translation Layer) support
< >  NAND SSFDC (SmartMedia) read only translation layer
< >  Log panic/oops to an MTD buffer
      RAM/ROM/Flash chip drivers --->
      Mapping drivers for chip access --->
      Self-contained MTD device drivers --->
<*>  NAND Device Support --->
< >  OneNAND Device Support --->
      UBI - Unsorted block images --->

      <Select>    < Exit >    < Help >
```

```
.config - Linux Kernel v2.6.28-pragmatec Configuration

----- NAND Device Support -----
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

-- NAND Device Support
[ ]   Verify NAND page writes
[ ]   NAND ECC Smart Media byte order
[ ]   Enable chip ids for obsolete ancient NAND devices
< >  GPIO NAND Flash driver
<*>  NAND Flash support for S3C2410/S3C2440 SoC
[ ]   S3C2410 NAND driver debug
[ ]   S3C2410 NAND Hardware ECC
[ ]   S3C2410 NAND IDLE clock stop
< >  DiskOnChip 2000, Millennium and Millennium Plus (NAND reimp
v(+)
```

<Select> < Exit > < Help >

- Entrées spéciales dans `/dev` pour chaque « partition » de flash
 - Une entrée en mode caractère pour la configuration (effacement, écriture, ...) => `/dev/mtdX`
 - Une entrée en mode bloc pour l'utilisation avec un système de fichiers => `/dev/mtdblockX`
- Une entrée dans `/proc` pour la liste des partitions

```
# cat /proc/mtd
```

```
dev:      size    erasesize  name
mtd0: 00030000 00004000 "bootloader"
mtd1: 00300000 00004000 "Kernel"
mtd2: 00500000 00004000 "rootfs"
mtd3: 03700000 00004000 "userland"
```

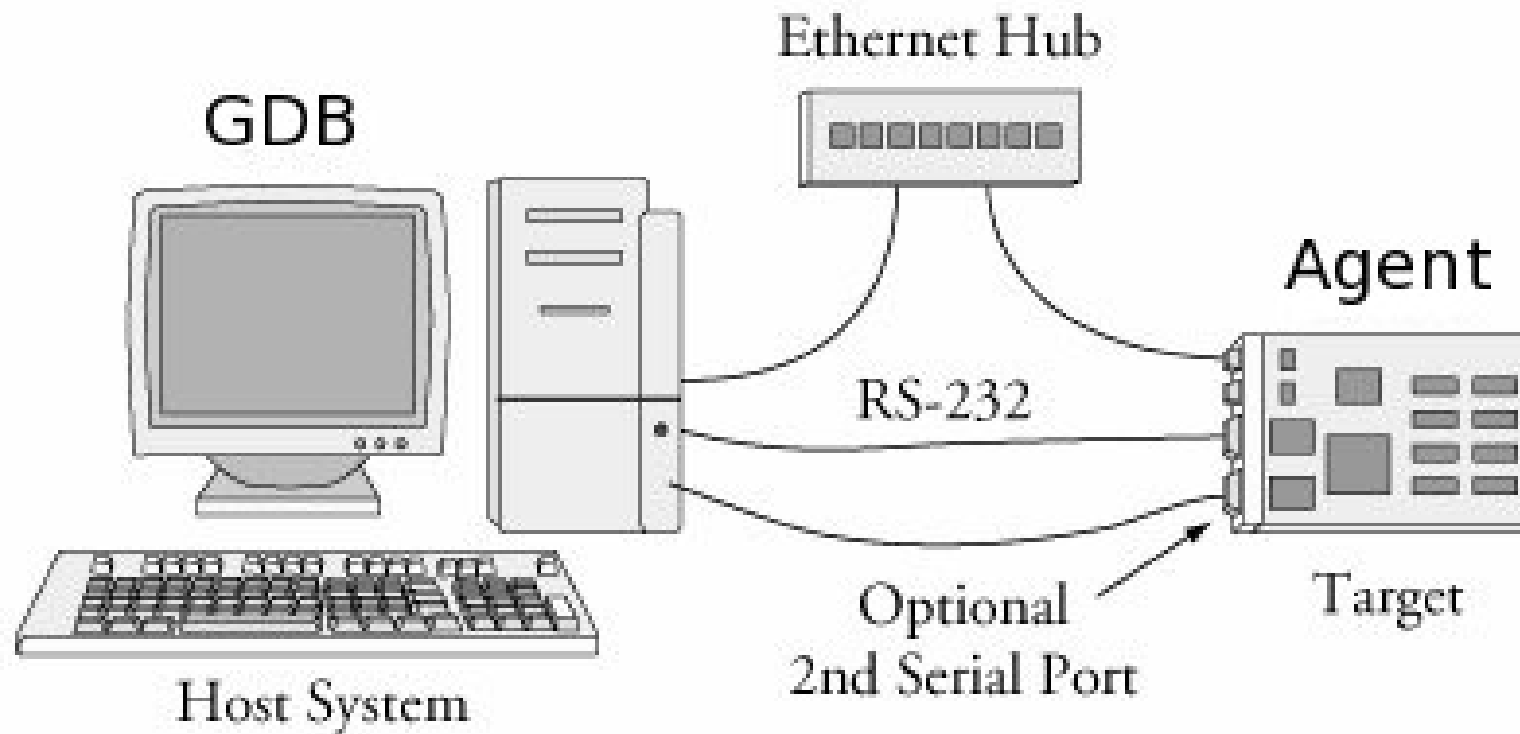
- On peut manipuler la mémoire flash depuis Linux grâce à *mtd-utils* (si la partition n'est pas montée !!)
- Le nom du fichier spécial `/dev/mtdX` est indépendant du type de flash (NOR ou NAND)
- Principales commandes :
 - # `flash_eraseall [-j] /dev/mtdX`
 - # `nandwrite -p /dev/mtdX rootfs.jffs2` (si NAND)
 - # `cat rootfs.jffs2 > /dev/mtdX` (si NOR)

Mise au point

- syslog: indispensable au suivi correct des traces d'un système. Version améliorée avec rsyslog (serveur de traces, SQL, ...)
- Valgrind: exécution du programme dans une « machine virtuelle » → Pas de re-compilation mais performances dégradées
- strace / ltrace: trace des appels systèmes et bibliothèques. Rustique mais efficace, filtrage des appels nécessaires.
- GDB

- Mode distant (remote) basé sur un protocole standard « remote protocol » développé pour GDB
- Utilisation d'un agent sur la cible et d'un débogueur croisé sur l'hôte :
 - Agent gdbserver pour espace utilisateur
 - Pour l'espace noyau :
 - KGDB (option du noyau)
 - d'autres agents intégrés à des sondes JTAG (bdiGDB pour BDI/Abatron) ou des projets comme OpenOCD
- Ces outils peuvent être intégrés à des IDE (Eclipse, QtCreator, ...)

Mise au point à distance, principe



- Cas le plus simple pour le développement applicatif
- Fonctionne avec un lien Ethernet ou série RS-232 entre cible et hôte
- Exemple :

```
# gdbserver :9999 myprog
```


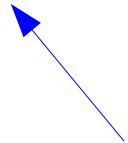
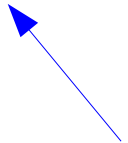
← Sur la cible

```
Process myprog created; pid = 12810
```

```
$ arm-none-linux-gnueabi-gdb myprog
```

← Sur l'hôte

```
GNU gdb (Sourcery G++ Lite 2010.09-50) ...  
...  
(gdb) target remote 192.168.0.30:9999  
Remote debugging using 192.168.0.12:9999
```

- Historiquement développé par LynsysSoft, repris par Wind River
 - Peu de mise à jour sur la version « libre »
 - Patch finalement accepté dans Linux « mainline » pour 2.6.26 => Kernel hacking/KGDB:
 - Utilisation de `vmlinux` (non compressé) sur le poste de développement
 - Options `bootargs` coté cible :
 - `kgdboc=ttySAC1,115200` 
 - `kgdboe=@192.168.0.1/,@192.168.0.2`
 - `kgdbwait`
-  Cible  Hôte

- ATTENTION: kgdboe n'est plus *mainline* en l'état
- Coté GDB hôte :

```
$ arm-none-linux-gnueabi-gdb vmlinux
```

```
...
```

```
(gdb) b sys_sync    → arrêt sur sync
```

```
(gdb) b panic
```

```
(gdb) target remote /dev/ttyS0
```

```
(gdb) c
```



Port série (hôte)

- Mise au point de modules .ko => cas le plus fréquent
- Insérer le module par modprobe ou insmod
- Obtenir les valeurs des sections :

```
$ cat /sys/module/helloworld/sections/.text  
0xbf000000
```

- Coté GDB, ajout des adresses des sections
(gdb) add-symbol-file helloworld.ko 0xbf000000
-s .data 0xbf00052e -s .bss 0xbf000660 -s
.rodata 0xbf0000ac
- Debug module_init() avec version spéciale de
GDB => *pending breakpoint*

```
(gdb) set solib-search-path chemin-accès-modules  
(gdb) b my_module_init  
(gdb) Breakpoint 1 (my_module_init) pending.
```

Utilisation d'un outil de production

Créer une distribution embarquée ?

- Produit d'éditeur (Wind River, MV, ...) → €€€
- Utiliser un outil de génération : Buildroot, OpenEmbedded, OpenWrt, LTIB
- Adapter une distribution Linux classique
 - Limité au niveau matériel
 - Empreinte mémoire importante
 - Cas très particulier (x86, Debian/ARM)
- Créer la distribution « from scratch »
 - Complexe (à faire une fois !)
 - Difficile/impossible à industrialiser:
 - gestion des dépendances
 - spécificités de la cible
 - évolutions



- LA solution « libre » de création de distribution
- Un « moteur » crée la distribution à partir des sources des composants adaptés en appliquant des patch
- Ne fournit pas les sources: uniquement les patch et les règles de production prenant en compte les dépendances :-)
- Peut produire la chaîne croisée
- Produit les différents éléments de la distribution
 - Image du bootloader (u-boot.bin)
 - Noyau Linux (zImage, uImage)
 - Image du root-filesystem (rootfs.jffs2, ...)
- Attention, nous parlons d'outil de *construction/intégration* et non pas de *développement*

Les principaux outils disponibles

- OpenEmbedded
 - Moteur écrit en Python
 - Très puissant mais (très) lourd
 - Basé sur des fichiers de configuration (?)



- Buildroot
 - Basé sur la commande make
 - Au départ un démonstrateur pour uClibc
 - Désormais un véritable outil, bien maintenu !



- OpenWrt
 - Dérivé de BR
 - Orienté vers les IAD (Internet Access Device)
 - Gère les paquets binaires



Autres: LTIB (Freescale), PTXdist (Pengutronix)

- <http://buildroot.uclibc.org/docs.html>
- <http://elinux.org/images/2/2a/Using-buildroot-real-project.pdf>
- <http://www.delafond.org/traducmanfr/deb/man1/fakeroot.1.html>
- <http://lwn.net/Articles/330985>
- <https://openwrt.org/>
- <http://www.openembedded.org>
- Chapitres 11 et 15 de l'ouvrage *Linux embarqué, 4ème édition* sur <http://www.editions-eyrolles.com/Livre/9782212134827/linux-embarque>
- Démonstration OE sur carte Eukréa sur <http://www.youtube.com/watch?v=5VPB8LeCloM>
- <http://www.linuxembedded.fr/2011/08/ajouter-un-package-dans-openembedded-en-5-minutes>