open wide INGENIERIE

JDEV2013

Le noyau Linux

Pierre Ficheux (pierre.ficheux@openwide.fr)

Septembre 2013

Sujets traités

- Principes du noyau, espaces noyau et utilisateur
- Histoire
- Principaux concepts
- Nommage des versions
- Compilation d'un noyau Linux standard

Introduction

- Le noyau (Linux) est un composant fondamental du système
- Le système est constitué :
  - d'un noyau permettant l'accès au matériel → *kernel space*
  - d'un ensemble d'utilitaires et bibliothèques permettant d'y accéder par des « appels systèmes » → *user space*
- Sous GNU/Linux, l'accès au noyau est *direct* → `open()`, `read()`, `write()`, `close()`
- Le noyau caractérise le système
  - API de développement noyau spécifique
  - Standardisation assurée par les appels systèmes → Mac OS X et GNU/Linux utilisent les composants GNU

Noyau Linux, principe

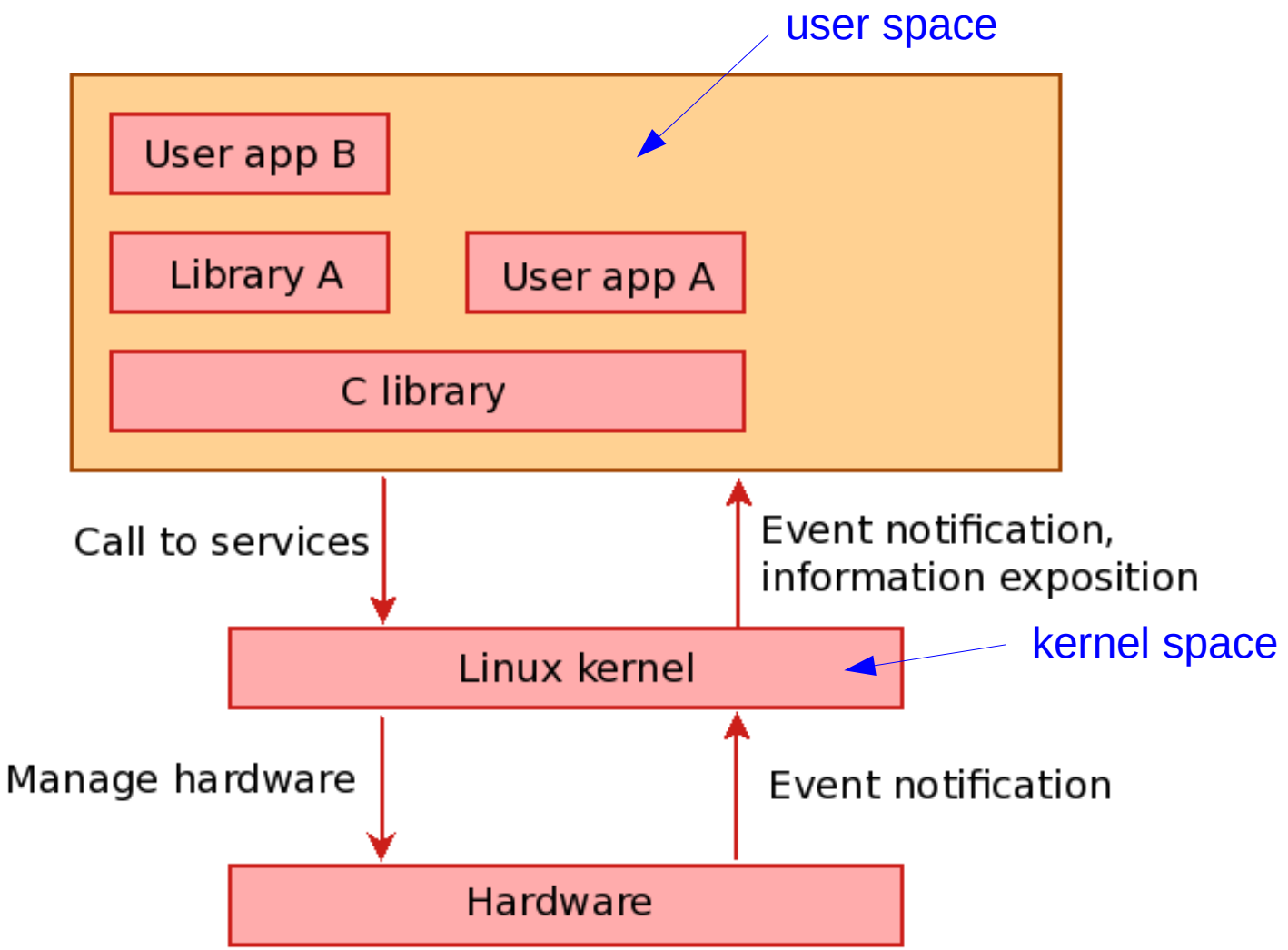


Schéma M. Ripard

Les origines de Linux

- Développé par Linus Torvalds en 1991 afin d' « écrire un noyau POSIX libre »
- Aucune vocation industrielle, juste un « hobby » d'étudiant
- 50 Ko de sources pour la première version 0.01, près de 90 Mo aujourd'hui (15 M lignes de codes)
- Linux est lié au projet GNU (GNU is Not UNIX) de Richard M. Stallman (MIT, années 80) et initialement à MINIX
- Le nom officiel de l'OS Linux est *GNU/Linux* car *Linux* correspond *uniquement* à la partie noyau
- Internet fortement contribué au succès de Linux
- Pour un non initié, la personnalité de Linus Torvalds est plus « rassurante » que celle de R. Stallman :-)

open wide
INGENIERIE

Les « parents » de GNU/Linux

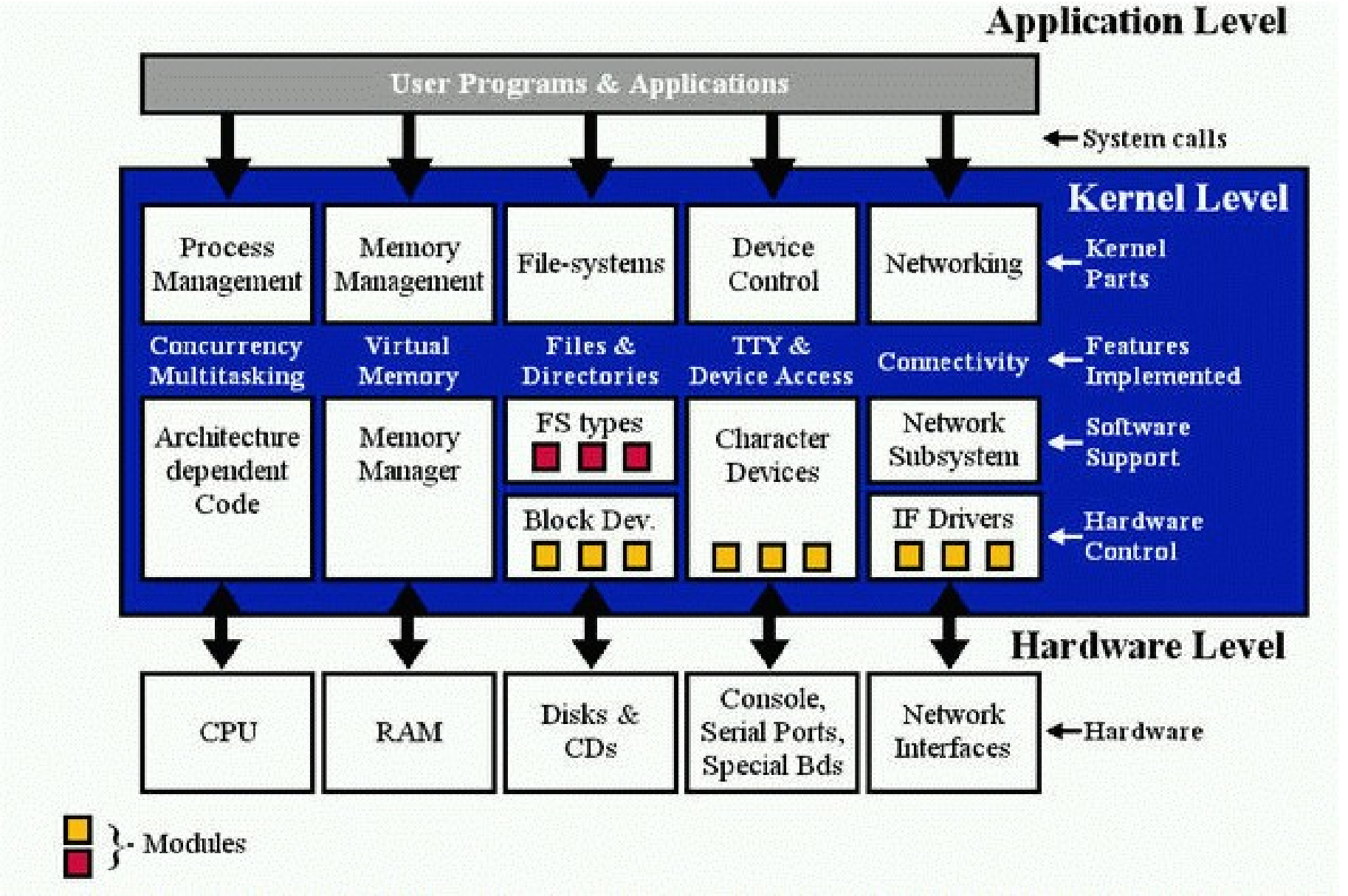
Les concepts de Linux

- Noyau *monolithique* (un seul fichier exécutable) + modules dynamiques (`.ko`)
- Seul le noyau et ses modules occupent l'espace noyau
- Licence GPL v2 depuis 1992
- Création des processus via `fork()` et `exec()`
- Multi-threading depuis fin 2003 (2.6)
- Nombreuses piles réseau (IPv4, IPv6, Ethernet, etc.)
- Très large communauté
- La gouvernance du projet est entre les mains de Linus Torvalds → noyau standard, « vanilla » ou « mainline »
- De nombreux « fork » du noyau (Android, BSP constructeurs, ...)

Les concepts de Linux, suite.

- Portabilité, 26 architectures supportées (x86, arm, …)
  → répertoire `arch`
- « Scalabilité » du super calculateur à l'embarqué
- Conformité aux standards, interopérabilité
- Très bon support réseau (depuis 2.6)
- Sécurité « auditée » en permanence
- Stabilité → concours d' « uptime » :-)
- Modularité, configuration en fonction des besoins
- Outil de configuration + ou - « graphique »
- « Simple » à programmer, nombreux exemples et documentations
- Documentation « en ligne » dans le répertoire `Documentation` des sources

Architecture du noyau



Nommage des versions

- Jusqu'à 2.6 (exclu)
  - versions stables paires : `1.0.x`, `2.2.x`, `2.4.x`
  - versions instables impaires : `2.5.x`
- Version 2.6 (2003/2011) :
  - 4 chiffres au lieu de 3 : `2.6.12.x`, `2.6.30.x`
  - plus de branche instable
- Version 3.x (depuis juillet 2011)
  - branches officielles : `3.x` (`3.0`, `3.1`, ..., `3.10`)
  - corrections : `3.x.y` (`3.2.21`)
- Noyaux LTS (*longterm*), maintenance durant plusieurs années : `2.6.32`, `2.6.34`, `3.0`, `3.2`, `3.4`, ...

Pourquoi compiler le noyau?

- Compilation « croisée » (x86 → ARM), cas d'Android
- Compatibilité avec les « patch » (temps-réel, etc.)
- Optimisation en empreinte mémoire et performances
- Maîtrise du fonctionnement

Compilation du noyau « mainline »

- Obtenir les sources sur http://www.kernel.org
- Extraire les sources dans un répertoire de travail
- Configurer le noyau par (au choix) :

```
$ make menuconfig ou xconfig ou gconfig
$ make <myboard>_defconfig
$ cp <path>/config.kernel .config ; make oldconfig
```

- Compiler par `make [bzImage]`

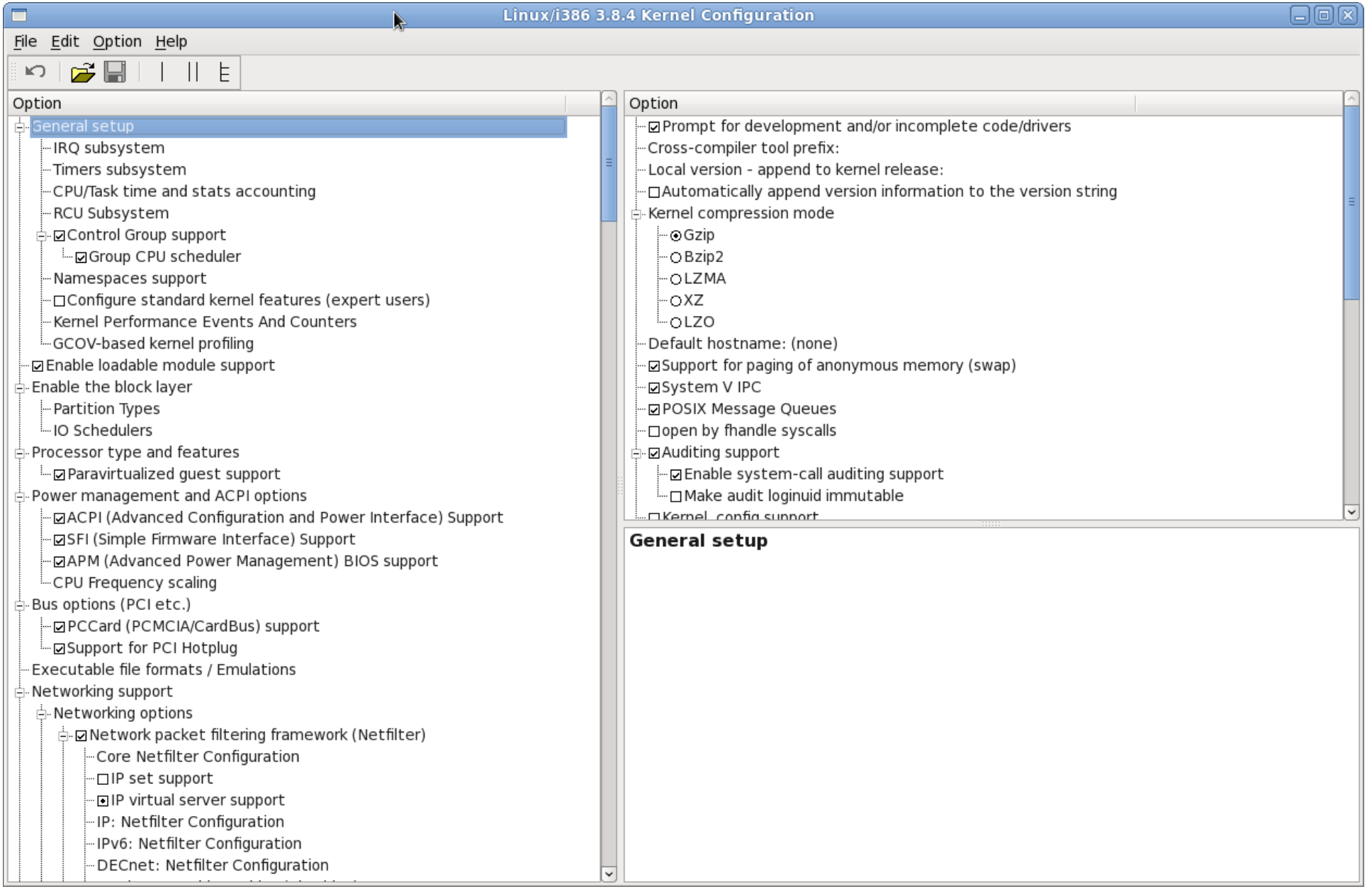
```
→ arch/<arch>/boot/[b]zImage + modules (.ko)
```

- Installer (optionnel) :

```
$ sudo make modules_install [INSTALL_MOD_PATH=<path>]
$ sudo make install [INSTALL_PATH=<path>]
```

open wide INGENIERIE

make xconfig



Compilation croisée

- Même procédure que la compilation « native »
- On utilise en plus les variables
  - `ARCH` → nom de l'architecture noyau (x86, arm, ...)
  - `CROSS_COMPILE` → préfixe du compilateur
- Ajout des variables d'environnement

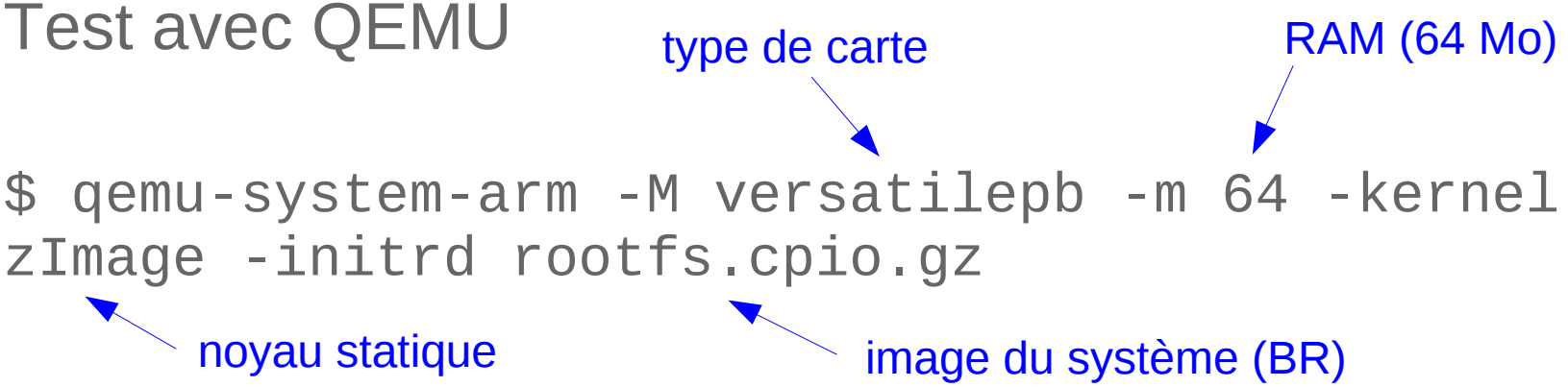

```
$ export PATH=$PATH:$HOME/arm-2012.03/bin
$ export ARCH=arm
$ export CROSS_COMPILE=arm-none-linux-gnueabi-
```

- On place ces lignes dans un script à « sourcer »
- Test de la chaîne de compilation :

```
$ source <path>/set_env_sourcery.sh
$ arm-none-linux-gnueabi-gcc -v
$ arm-none-linux-gnueabi-gcc -o hello hello.c
```

Compilation d'un noyau ARM

- On utilise l'architecture émulée Versatile PB (ARM9)
- Carte ancienne mais très bon support dans QEMU
- Chargement de l'environnement
- Chargement de la configuration du noyau
- Compilation du noyau statique → `zImage`
- Test avec QEMU



Références

- http://www.kernel.org
- http://wiki.kernelnewbies.org/LinuxChanges
- http://lwn.net
- http://lwn.net/Kernel/LDD3
- http://elinuxdd.com