# NMV :
# Installer un nouveau noyau linux
## Version 1.05

Julien Sopena[1]

[1]julien.sopena@lip6.fr
Équipe REGAL - INRIA Rocquencourt
LIP6 - Université Pierre et Marie Curie

Master SAR 2ème année - NMV - 2017/2018

# Grandes lignes du cours

Étape 1 : Récupération du code source et des patch
Étape 2 : Vérification de l'intégrité des sources téléchargées
Étape 3 : Application des patch sur le code source
Étape 4 : Configuration du noyau
Étape 5 : Compilation du noyau et des modules
Étape 6 : Installation du noyau
Étape 6 : Installation du noyau

# Outline

# Étape 1 : récupération des sources officielles

Les sources du kernel sont accessibles depuis le site : **https ://kernel.org**

```
wget https://kernel.org/pub/linux/kernel/v3.x/linux-3.17.3.tar.xz
wget https://kernel.org/pub/linux/kernel/v3.x/linux-3.17.3.tar.sign
```

Il peut être aussi nécessaire de récupérez une mise à jour (ensemble de correctifs) pour la version x.y.<z-1> :

```
wget https://kernel.org/pub/linux/kernel/v3.x/patch-3.17.3.xz
wget https://kernel.org/pub/linux/kernel/v3.x/patch-3.17.3.sign
```

Mais le mieux reste d'utiliser le dépôt git sur **git.kernel.org** :-)

```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
```

# Outline

# Cette vérification est une étape nécessaire

## Attention

Le noyau est le code le plus critique du système, il est donc indispensable de vérifiez l'intégrité des sources.

C'est très simple en utilisant la signature gpg

Mais encore plus avec git :-)

# Vérification de l'intégrité des sources

1. On commence par décompresser l'archive car la signature est sur le tar

```
unxz linux-3.17.3.tar.xz
```

# Vérification de l'intégrité des sources

1. On commence par décompresser l'archive car la signature est sur le tar

```
unxz linux-3.17.3.tar.xz
```

2. Puis on peut faire la validation avec *gpg*

```
gpg -verify linux-3.17.3.tar.sign
  gpg: Signature made Fri Nov 14 19:11:14 2014 CET using RSA key ID 6092693E
  gpg: Can't check signature: No public key
```

# Vérification de l'intégrité des sources

1. On commence par décompresser l'archive car la signature est sur le tar

```
unxz linux-3.17.3.tar.xz
```

2. Puis on peut faire la validation avec *gpg*

```
gpg -verify linux-3.17.3.tar.sign
  gpg: Signature made Fri Nov 14 19:11:14 2014 CET using RSA key ID 6092693E
  gpg: Can't check signature: No public key
```

3. On doit charger la clé public gpg qui a servie pour la signature

```
gpg -keyserver hkp://keys.gnupg.net -recv-keys 6092693E
  ...
  gpg: key 6092693E: public key "Greg Kroah-Hartman <greg@kroah.com>" imported
  ...
```

# Vérification de l'intégrité des sources

1. On commence par décompresser l'archive car la signature est sur le tar

```
unxz linux-3.17.3.tar.xz
```

2. Puis on peut faire la validation avec *gpg*

```
gpg -verify linux-3.17.3.tar.sign
  gpg: Signature made Fri Nov 14 19:11:14 2014 CET using RSA key ID 6092693E
  gpg: Can't check signature: No public key
```

3. On doit charger la clé public gpg qui a servie pour la signature

```
gpg -keyserver hkp://keys.gnupg.net -recv-keys 6092693E
  ...
  gpg: key 6092693E: public key "Greg Kroah-Hartman <greg@kroah.com>" imported
  ...
```

4. Maintenant on peut vraiment valider les sources

```
gpg -verify linux-3.17.3.tar.sign
 gpg: Signature made Fri Nov 14 19:11:14 2014 CET using RSA key ID 6092693E
  gpg: Good signature from "Greg Kroah-Hartman <greg@kroah.com>" [unknown]
```

# Outline

# C'est quoi un patch ?

**Definition**

Un **patch** n'est que le résultat d'un **diff** enregistré dans un fichier.
Il permet de passer d'une version à l'autre.

- **diff** : Commande qui compare des fichiers ligne par ligne
  - indique les lignes ajoutées ou supprimées ;
  - peut ignorer les casses, les tabulations, les espaces ;
  - option `-u` pour créer des patchs unifiés, avec plus d'informations.
- **patch** : Commande qui permet d'appliquer une liste de différences.

```
diff toto-new.c toto-orig.c > correction.patch
xz correction.patch
```

```
xzcat correction.patch.xz | patch toto-orig.c
```

# Application des patchs (si nécessaire)

La commande patch peut appliquer un ensemble de changements sur une arborescence de fichiers sources.

Dans ce cas l'option **-p <n>** permet de tronquer <nb> répertoires dans les noms de fichiers

Pour patcher les sources d'un noyau Linux :

- toujours à appliquer sur la version x.y.<z-1>
- les patchs sont prévus pour n=1 : `patch -p1 < patch`

```
cd /usr/src/linux
xzcat patch-3.18.xz | patch -p1 --dry-run
```

# Outline

# Configuration du nouveau noyau

**Definition**

La **configuration** d'un noyau, consiste à définir quelles fonctionnalités y seront intégrées et le cas échéant si elles le seront statiquement ou sous forme de module. Par défaut le `Makefile` utilise le fichier **.config**

Plusieurs méthodes peuvent être utilisées pour éditer la configuration :

**vim .config :** édition de la configuration à la main.

**make config :** interface en mode texte

**make menuconfig :** interface utilisant ncurses

**make xconfig :** interface graphique utilisant *Qt*

**make gconfig :** interface graphique utilisant *Gtk*

# Configuration de départ

Pour construire une nouvelle configuration, on peut partir :

- **d'une configuration minimum (aucune option) :**

```
make allnoconfig
```

- **de la configuration par défaut :**

```
make alldefconfig
```

- **de la configuration actuelle :**
  1. certaines distributions la placent dans `/boot`

```
cp /boot/config-`uname -r`* .config
```

  2. si le noyau actuel a été compilé avec l'option `CONFIG_IKCONFIG_PROC=Yes`

```
zcat /proc/config.gz > .config
```

# Modifier une configuration

Pour vous aidez à modifier votre configuration, vous pouvez :

- **faire une mise à jour interactive :**

```
make oldconfig
```

- **faire une mise à jour automatique :**

```
make silentoldconfig
```

- **désactiver les modules actuellement déchargés :**

```
make localmodconfig
```

- **intégrer statiquement les modules actuellement chargés :**

```
make localyesconfig
```

# Connaître son matériel

Pour connaître son matériel, on peut utiliser :

- `lshwd`, `lspci`, `lsusb`, ...
- `dmidecode`
- `hdparam`
- `cat /proc/cpuinfo`, `cat /proc/meminfo`, ...
- Et surtout `dmesg`

# Numéroter son noyau

La version du noyau est définie par des variables du Makefile :

1. son numéro de version <VERSION>.<PATCHLEVEL>.<SUBLEVEL>
2. la variable <EXTRAVERSION> permet de distinguer des images compilées à partir des mêmes sources

```
VERSION = 4
PATCHLEVEL = 2
SUBLEVEL = 3
EXTRAVERSION = -rc2
```

Ces informations sont accessibles lorsque le noyau sera chargé

```
uname -r
      4.2.3-rc2
```

# Personnaliser la version du noyau

Pour personaliser le numéro de version du noyau, on peut :

1. **modifier le `Makefile` :** mais alors le dépot n'est plus à jour
2. **modifier le `.config` :** grâce à l'option **Local version**

Dans la section *General setup*, modifiez l'option de configuration

```
General setup  --->
  (-sopena) Local version - append to kernel release
```

Après recompliation du noyau, on observe le résultat avec `uname`

```
uname -a
      Linux vm-sopena linux-4.2.3-sopena #1 SMP Sat Mar 21
      13:40:39 CET 2015 x86_64 GNU/Linux
uname -r
      linux-4.2.3-sopena
```

# Outline

# Compilation du noyau et des modules

La compilation est la phase la plus simple mais aussi la plus longue :

*80 minutes pour un noyau 3.18 sur unXeon E5-1603 2.80 GHz.*

```
time make
   real 80m15.486s
   user 74m54.606s
   sys   5m32.300s
```

Pour écourter l'attente on peut paralléliser la compilation avec l'option `--jobs` de make qui fixe le nombre de tâches lancer simultanément.

```
cat /proc/cpuinfo | grep processor | wc -l
  4
time make -j 5
   real 21m34.334s
   user 75m38.123s
   sys   4m58.757s
```

# Fichiers générés après la compilation

**./vmlinux :** Image brute du noyau Linux, non compressée

⇒ ce fichier ELF est réservé au debogage et profilage

**./System.map :** Fichier contenant la table des symboles du noyau

⇒ ce fichier n'est pas nécessaire mais utile au debogage

**./arch/<arch>/boot/bzImage :** Image du noyau compressée avec zlib

⇒ c'est cette version que l'on préfèrera charger en mémoire

```
du -sh vmlinux arch/x86/boot/bzImage
  13M vmlinux
  3.9M arch/x86/boot/bzImage
```

La commande file permet d'extraire des informations de ces images :

```
file arch/x86/boot/bzImage
   arch/x86/boot/bzImage: Linux kernel x86 boot executable bzImage, version
   3.18.0-rc4-ARCH (sopena@ari) #2 SMP PREE, RO-rootFS, swap_dev 0x3, Normal VGA
```

# Outline

# Installation du noyau

L'installation du noyau compilé se fait en trois étapes :

1. Installation du noyau et du mapping dans `/boot` (par défaut)

```
make install
```

# Installation du noyau

L'installation du noyau compilé se fait en trois étapes :

1. Installation du noyau et du mapping dans `/boot` (par défaut)

```
make install
```

2. Installation des modules dynamiques dans `/lib/modules/xx/kernel` et dans `/lib/modules/xx/extra` s'ils sont extérieurs aux sources

```
make modules_install
```

# Installation du noyau

L'installation du noyau compilé se fait en trois étapes :

1. Installation du noyau et du mapping dans `/boot` (par défaut)

```
make install
```

2. Installation des modules dynamiques dans `/lib/modules/xx/kernel` et dans `/lib/modules/xx/extra` s'ils sont extérieurs aux sources

```
make modules_install
```

3. Installation des firmwares dans `/lib/modules/xx/kernel/lib/firmware`

```
make firmware_install
```

# Outline

# Fichiers installés après un make install

Les fichiers du noyau sont installés dans le répertoire `/boot/` :

- **/boot/vmlinuz-<version> :** Image du noyau
- **/boot/System.map-<version> :** Stocke les adresses des symboles (primitives systèmes) du noyau
- **/boot/initrd-<version>.img :** Initial RAM disk, contenant les modules nécessaires pour monter le système de fichier root (`make install` lance `mkinitrd`).

L'installation peut aussi mettre à jour la configuration du *bootloader* :

- **/etc/grub.conf ou /etc/lilo.conf :** make install met à jour les fichiers de configuration de votre bootloader pour supporter votre nouveau noyau. Il relance /sbin/lilo si LILO est votre bootloader.

# Symboles des primitives du noyau

```
cat /boot/System.map
  00100000 A phys_startup_32
  bfffe400 A __kernel_vsyscall
  bfffe410 A SYSENTER_RETURN
  bfffe420 A __kernel_sigreturn
  bfffe440 A __kernel_rt_sigreturn
  c0100000 A _text
  c0100000 T startup_32
  c01000a4 T startup_32_smp
  c0100124 t checkCPUtype
  c01001a5 t is486
  c01001ac t is386
  c0100210 t L6
  c0100212 t check_x87
  c010023a t setup_idt
  c0100257 t rp_sidt
  (...)
```