S. Lavirotte
Polytech'Nice – Sophia
Si5 – Master 2 IFI
2015-2016

TD n° 2
Noyau Linux et Appels Systèmes

Le but de ce TD est de vous familiariser avec le noyau Linux. Vous commencerez par configurer et compiler un noyau avant d'y ajouter des fonctionnalités afin de découvrir les appels systèmes.

1.1 Installation du Noyau Linux

1.1.1 Utilisation d'un noyau préinstallé sur une image

Si vous ne l'avez pas déjà fait, récupérez à l'adresse suivante une image disque contenant les sources du noyau pour ce TD :

```
https://trolen.polytech.unice.fr/cours/sae/td01/sdb-linux-kernel.7z
```

Montez cette image dans l'arborescence de votre système de fichier dans `/work` :

```
mount /dev/sdb1 /work
```

Vous avez maintenant accès dans `/work` aux sources de noyau Linux (un alias `mwork` a été créé pour faire cette commande).

1.1.2 Récupération du noyau

Dans le cas où vous auriez besoin d'installer vous-même votre noyau à partir de sources qui se trouvent sur le site http://www.kernel.org/, il vous faudra vérifier que l'archive fournie avec la clé de `kernel.org` qui a le numéro `0x6092693E`:

```
gpg --keyserver hkp://keys.gnupg.net --recv-keys 6092693E
gpg --verify linux-2.6.32.68.tar.sign linux-2.6.32.68.tar
```

et à installer les sources dans `/usr/src`.

1.2 Familiarisation avec les sources du noyau

En tant que développeur de modules pour Linux, vous aurez très souvent à trouver où se trouve telle ou telle fonctionnalité dans les sources. Vous devrez être capable de naviguer dans plus de 420Mo de sources et de documentation, soit plus de `12.500.000` de lignes de code C et documentation... Il est donc important de se « familiariser » avec l'organisation des sources du noyau !

Vous pourrez vous aider des vos outils habituels comme `locate`, `find`, `grep`, `ctags`, ...

Vous pouvez vous amuser à chercher :

- le code source du driver pour ne2k (la carte émulée de notre machine virtuelle)
- la documentation sur les cartes TV DVB (Digital Video Broadcast) supportées par le noyau
- ...

2 Configuration et Compilation du noyau

2.1 Configuration du noyau

Nous souhaitons construire un noyau Linux pour une configuration aux ressources limitées. Nous devons donc inclure dans cette configuration le minimum de pilotes afin de réduire au minimum l'emprunte mémoire du noyau que nous utiliserons (au passage, cela gagnera aussi énormément de temps de compilation).
Vous partirez d'une version minimale (`make allnoconfig`) du noyau et ajouterez les pilotes au besoin (sous forme de module ou build-in). Ne décochez aucune option sur cette configuration de base sous peine de produire un noyau non fonctionnel.
Comme nous sommes sous console, pour configurer le noyau, vous utiliserez `make menuconfig` and installant auparavant le paquetage `libncurses-dev`.

S. Lavirotte
Polytech'Nice – Sophia
Si5 – Master 2 IFI
2015-2016

TD n° 2
Noyau Linux et Appels Systèmes

Le but de ce travail est de vous faire créer un noyau très petit qui se compile très vite et qui vous fournisse un support pour les TD suivants. Vous veillerez à activer les fonctionnalités suivantes sur votre noyau (en tant que fonction incluse dans le noyau et non en tant que module).
Pour pouvoir faire une configuration correcte des fonctionnalités dans le noyau, il est nécessaire de connaître la configuration matérielle pour laquelle votre noyau est compilé. Dans notre cas, la configuration matérielle est celle fournie par la machine virtuelle. Voici donc une courte description de celle qui vous est fournie :

- Microprocesseur : c'est celui de votre machine physique car VirtualBox est un virtualiseur
- Mémoire : 256Mo
- Disques durs SATA (avec contrôleur de disques SATA AHCI) avec formatage des partitions en ext3
- CD-ROM : IDE (utilise soit votre lecteur physique, soit une image iso)
- Carte réseau : pcnet32 ou intel e1000
- Contrôleur USB : USB 2.0

Il est donc nécessaire d'activer les drivers dans le noyau afin de pouvoir communiquer avec ces matériels :

Configuration générale du noyau pour un PC x86 (processeur, mémoire, réseau, input classiques)

General setup --->
  [*] Support for paging of anonymous memory (swap)
  [*] System V IPC
  () Kernel log buffer size (16Ko)
  [*] Optimize for size
[*] Enable loadable module support --->
  [*] Module unloading
Processor type and features --->
  [*] Symmetric multi-processing support
  [*] Single-depth WCHAN output
    Processor Family (lire la documentation et voir le numéro `model name` et `cpu_family` dans `/proc/cpuinfo`)
  [*] Generic x86 support
  [*] Enable x86 board specific fixups for reboot
  High Memory Support (off)
  [*] Reserve low 64K of RAM on AMI/Phoenix BIOSen
Power Management and ACPI options --->
  [*] Power Management support
  [ ] Suspend to RAM and standby
  [*] CPU idle PM support
Bus options (PCI etc.) --->
  [*] PCI support
Executable file formats / Emulations --->
  [*] Kernel support for ELF binaries
Networking support --->
  Networking options --->
    <*> Packet socket
    <*> Unix domain sockets
    [*] TCP/IP networking
    Pas de support IPSec
    Pas de support IPv6
  [ ] Wireless
Device Drivers --->

```
  [*] Block devices --->
    <*> Loopback device support
    <*> RAM block device support
  Input device support --->
    <*>Generic input layer (needed for keyboard, mouse ...)
    [*] Provide legacy /dev/psaux device
    [*] Event Interface
    Mice --->
      <*> PS/2 mouse (mais pas les sous-sections)
  Character devices --->
    [*] Support for binding and unbinding console drivers
  [*] HID Devices --->
    <*> Generic HID support
  [*] USB Support
    <*> Support for Host-side USB
    <*> UHCI HCD (most Intel and VIA)
File systems --->
  <*> Second extended fs support
  <*> Ext3 journaling file system support
  [*] Dnotify support
  [*] Inotify file change notification support
  [*] Inotify support for userspace
  <*> FUSE (Filesystem in Userspace) support
  Pseudo filesystems --->
    [*] Virtual memory file system support (former shm fs)
  * Native language support --->
    <*> NLS ISO 8859-1 (Latin 1; Western European Languages)
Kernel Hacking --->
      [*] Compile the kernel with frame pointers
  [*] Enable verbose x86 bootup info message
Library routines --->
  <*> CRC32c (Castagnoli, et al) Cyclic Redundancy-Check
```

Configuration pour le support matériel spécifique dans la machine virtuelle

Support SCSI

```
Enable the block layer --->
  [*] Block layer SG support v4
Device Drivers  --->
  SCSI device support  --->
      <*> SCSI device support
      <*> SCSI disk support
      <*> SCSI CDROM support
      <*> SCSI generic support
  [*] SPI support
```

Support SATA

Device Drivers --->
  <*> Serial ATA and Parallel ATA Drivers (libata) --->
    <*> AHCI SATA support

Contrôleurs réseaux

Device Drivers --->
  [*] Network device support --->
    [*] Ethernet (10 or 100 Mbit) --->
      [*] EISA, VLB,PCI and on board controllers
      <*> AMD PCnet32 PCI support
      <*> PCI NE2000 and clones support (see help)
    [*] Ethernet (1000 Mbit) --->
      <*> Intel PRO/1000 Gigabit Ethernet support

En quittant la configuration, le programme vous demande de confirmer l'enregistrement de votre configuration. Cette configuration sera enregistrée dans le fichier `.config`. Si vous souhaitez faire une sauvegarde de ce fichier de configuration, il vous suffit de copier ce fichier sous un autre nom. Mais le `Makefile` du noyau prendra toujours le fichier `.config` comme étant le fichier de référence par rapport auquel il fera la compilation.

Et n'oubliez pas non plus d'activer la bonne option à `make` pour avoir une compilation le plus rapide possible :

```
make -j 2 x nb cœurs
```

2.2 Déploiement et test du noyau sur votre système local

Une fois la compilation terminée, il ne vous reste plus qu'à installer ce nouveau noyau sur votre système :

```
make install
```

Modifiez le chargeur de noyau qui est installé sur votre système (`/boot/grub/grub.cfg`) afin de vous proposer ce nouveau noyau au prochain reboot avec les bons paramètres.

Attention ! La dénomination de la partition root de votre système utilise le système de nommage UUID. Celui-ci ne peut être utilisé dans notre cas présent (utilisation via un script dans un système de fichier alternatif pour le boot que l'on verra lors d'un prochain cours). Vous devrez donc remplacer dans le fichier de configuration de grub la définition de la partition root du système de fichier en remplaçant l'UUID par `root=/dev/sda1`

Supprimez la référence à l'UUID pour la remplacer par `/dev/sda1` et supprimez l'option `quiet` pour le noyau `2.6.32.68`.

3 Appels système

3.1 Ajouts d'appels systèmes au noyau

Toujours sur votre noyau `2.6.32.68`, nous allons tenter d'ajouter un appel système et de l'invoquer par un programme de test.

Pour bien comprendre le fonctionnement d'un appel système, il suffit d'en créer un. Pour être sûr que vous avez bien compris, vous allez en créer deux nouveaux.
En prenant modèle sur l'exemple donné en cours, ajoutez les appels système suivants au noyau Linux :

```
void addition(int x, int y, int *res) ;
void increment(int *x) ;
```

Une fois ces nouveaux appels systèmes ajoutés au noyau, réamorcer la machine en prenant soin de démarrer sur le noyau auquel vous avez ajouté ces nouveaux appels système.

3.2 Programme testant les appels systèmes créés

Écrire un programme de test pour vos nouveaux appels systèmes (c'est le code qui devrait être ajouté à la bibliothèque C pour utiliser vos nouveaux appels système).
Ecrivez donc un simple programme (comme présenté en cours) pour faire ce test. Toutefois, lors de vos tests, vous prendrez soin d'essayer des adresses impossibles (ou protégées en écriture) pour le paramètre `res` qui est passé par adresse (NULL par exemple) et une constante pour le paramètre d'appel de la fonction `increment` (protection en écriture). Ainsi vous pourrez vraiment tester si vous avez correctement réalisé l'implémentation de ces appels systèmes en faisant correctement les copies de données de l'espace utilisateur vers l'espace noyau et vice versa).