TP3 : À la découverte du Filesystem

Blottière Paul <blottiere.paul@gmail.com>

Table of Contents

1. Généralités 1
    1.1. embsys 1
    1.2. Objectifs du TP 1
2. Exercice 1 : /boot, GRUB et initramfs 2
3. Exercice 2 : /dev, /sys et modules 4
4. Exercice 3 : /etc, udev et syslog 5

1. Généralités

1.1. embsys

L'ensemble des cours, exemples, PDF et TP est disponible sur le dépôt github https://github.com/pblottiere/embsys.

Si vous voulez cloner entièrement le dépôt :

```
$ git clone https://github.com/pblottiere/embsys
```

Si vous voulez cloner le dépôt mais avoir simplement les labs dans votre répertoire de travail :

```
$ git clone -n https://github.com/pblottiere/embsys --depth 1
$ cd embsys
$ git checkout HEAD labs
```

Si vous souhaitez mettre à jour un dépôt que vous avez préalablement cloné :

```
$ cd embsys
$ git pull
```

Si vous n'avez pas **git**, téléchargez le ZIP sur la page d'acceuil de **embsys**.

Le TP d'aujourd'hui se trouve ici : https://github.com/pblottiere/embsys/labs/3_fs_discovery.

1.2. Objectifs du TP

Dans le cadre de ce TP, nous allons partir à la découverte des éléments présents au sein du filesystem en parcourant :

- /boot
- /dev
- /etc
- /proc

- /sys
- /var

Nous allons réaliser des exercices afin de mettre en avant les circonstances dans lesquelles vous seriez susceptible d'avoir besoin de travailler dans ces répertoires.

Lors de ce TP, vous aurez les droits **root**. Vous devez donc être particulièrement **VIGILANT** à ce que vous faites ainsi qu'aux consignes des exercices afin de ne pas faire d'erreurs malencontreuses…

Il existe une règle simple pour éviter de telles erreurs : ne jamais être identifié root lorsque ce n'est pas nécessaire! Il ne s'agit donc pas ici d'être en root tout le long du TP…

2. Exercice 1 : /boot, GRUB et initramfs

En fonction des machines que vous rencontrerez, le répertoire /boot pourra contenir des éléments différents, mais un certain nombre sont relativement génériques :

- config-<KERNEL_VERSION>
- initrd.img-<KERNEL_VERSION>
- vmlinuz-<KERNEL_VERSION>

Dans le cas d'utilisation de GRUB comme bootloader, un répertoire /boot/grub est aussi présent.

Afin d'obtenir la valeur de <KERNEL_VERSION> :

```
> uname -r
4.2.0-1-amd64
> ls /boot/config-*$(uname -r)
/boot/config-4.2.0-1-amd64
```

Question 1 : *En se renseignant sur le net et grâce aux notions vues en cours, expliquez ce que sont les fichiers **config-**, **initrd.img-** et **vmlinuz-**.*

Question 2 : *En analysant le fichier /boot/grub/grub.cfg, déterminez quelle commande lance le kernel et quelle est celle qui charge l'initrd.*

En analysant le fichier /boot/grub/grub.cfg, on remarque que le kernel prend plusieurs paramètres.

Question 3 : *Expliquez l'utilité des paramètres **root**, **quiet** et **splash** passés au kernel.*

Dans le menu GRUB au démarrage de la machine, on peut, grâce à la touche **e**, rentrer dans la configuration du boot. De cette manière, on peut changer ponctuellement pour le prochain boot :

- les paramètres à passer au kernel
- l'initrd à charger en mémoire
- les messages à afficher au boot
- …

Question 4 : *Rebootez la machine et notez les phases de boot. Regardez le contenu du fichier /proc/cmdline.*

Question 5 : *Rebootez la machine et cette fois rentrez dans la configuration du grub grâce à la touche **e**. Modifiez les paramètres passés au kernel pour ne garder que **root=UUID=XXXXXXXXXXXX** et **ro**. Démarrez. Quel changements y a t-il eu par rapport au boot précédent? Qu'y a t-il dans le fichier /proc/cmdline?*

Nous allons maintenant étudier le fichier de configuration du kernel **/boot/config-<VERSION>**.

Question 6 : *À quel stade ce fichier de configuration est-il utilisé?*

Question 7 : *Comment voit on que le kernel Linux est un noyau modulaire seulement en analysant le fichier de configuration?*

Question 8 : *En étudiant le fichier de configuration du kernel, déterminez si l'interface sysfs est disponible pour les GPIO. Vérifiez.*

Depuis la version 2.6 de Linux, le fichier /boot/initrd.img- est une archive cpio compressée : c'est la notion d'initramfs. L'archive compressée chargée en RAM par le bootloader est ensuite décompressée par le kernel afin qu'un RFS minimal soit disponible en RAM. Ce RFS contient tout le nécessaire pour réaliser la phase de boot du kernel. Les avantages d'une telle technique sont :

- la possiblité de paramétrer le boot de manière simple
- l'initialisation du kernel dans un espace utilisateur minimal, permettant d'avoir accès à la libc ainsi qu'à des utilitaires de base

Une telle image peut être décompressée comme suit :

```
> gunzip -dc < initrd.img | cpio -id
```

Pour compresser :

```
> find ./ | cpio -H newc -o > ../custom.cpio
> cd ..
> gzip custom.cpio
> mv custom.cpio.gz custom.img
```

Question 9 : *Copiez l'initrd du /boot dans un répertoire de travail et décompressez la.*

Question 10 : *Où se trouve la libc dans le RFS décompressé? Quel type de libc est-ce (glibc, uclibc, elibc, …)?*

Dans le RFS décompressé, il existe un fichier **init** à sa racine. Ce fichier est exécuté par le kernel pour lancer la phase de démarrage, dont le premier processus (PID=1) /sbin/init.

Question 11 : *En analysant le fichier **init**, determinez comment les paramètres passés dans le grub sont récupérés et utilisés.*

Question 12 : *Modifiez le fichier **init** en rajoutant par exemple des print puis recompressez le RFS pour obtenir une initrd **custominitrd.img**. Grâce aux droits root, copiez **custominitrd.img** dans le répertoire /boot.*

Question 13 : *Rebootez la machine puis utilisez **e** pour rentrer dans le menu de configuration de grub. Modifiez les paramètres de boot pour utiliser **custominitrd.img** au lieu de l'image initiale et laissez seulement les paramètres kernel **root=…** et **ro**. Démarrez et observez vos messages lors de la phase de boot.*

3. Exercice 2 : /dev, /sys et modules

Comme vu dans les TP précédents, les ports séries sont disponibles à travers des fichiers spéciaux présents dans le /dev. Il existe de nombreux outils permettant de lire et écrire sur ces ports. Nous allons ici étudier l'outil minicom.

Dans un premier temps, compilez et exécutez le simulateur GPS de embsys :

```
> cd labs/gps
> make ok
> sh run.sh
```

Question 14 : *Quel est le type de périphérique (block ou char) d'un port série? Comment le vérifiez vous à travers le /dev?*

Question 15 : *Confirmez en étudiant le contenu du répertoire /sys/dev/.*

Question 16 : *Grâce à l'aide de minicom (**minicom -h**), connectez vous au port du simulateur et observez les trames passer.*

Question 17 : *Étudiez les possiblités de minicom à travers son menu de configuration et faites un log des trames dans un fichier minicom.log.*

Chaque fichier spécial du /dev est associé à un numéro majeur permettant au kernel de connaître le driver à utiliser. Par exemple :

```
> ls -l /dev/sfa[1-8]
v
brw-rw---- 1 root disk 8, 1 déc.  14 21:07 /dev/sda1
brw-rw---- 1 root disk 8, 2 déc.  14 21:07 /dev/sda2
brw-rw---- 1 root disk 8, 3 déc.  14 21:07 /dev/sda3
brw-rw---- 1 root disk 8, 4 déc.  14 21:07 /dev/sda4
brw-rw---- 1 root disk 8, 5 déc.  14 21:07 /dev/sda5
brw-rw---- 1 root disk 8, 6 déc.  14 21:07 /dev/sda6
brw-rw---- 1 root disk 8, 7 déc.  14 21:07 /dev/sda7
```

On voit ici que le numéro majeur correspondant au support de stockage est le 8. D'après la documentation kernel https://www.kernel.org/doc/Documentation/devices.txt et grâce à ce numéro, on retrouve le fait que le matériel sous jacent est bien un disque.

Question 18 : *Connectez le matériel distribué en TP et regardez l'influence dans le /dev. Que se passe t-il? Observez les messages kernel.*

Question 19 : *Déterminez le type de matériel à partir du numéro majeur et grâce à la documentation kernel.*

Le /sys, grâce au mécanisme de hotplug, est mis à jour par le kernel. Suite à cela, le kernel envoi des uevent dans l'espace utilisateur. Ces évènements sont écoutés par udev qui met alors à jour le /dev. On

peut notamment retrouver dans le /sys le driver utilisé par un matériel défini dans le /dev. Par exemple pour un disque :

```
> ls -l /dev/ttyS0
crw-rw---- 1 root dialout 4, 64 déc.  14 21:07 /dev/ttyS0
> ls /sys/dev/char/4:64/device/driver/
bind  serial8250  uevent  unbind
```

On voit ici que le driver utilisé se nomme serial8250.

Nous allons maintenant étudier les modules kernel. Ces modules sont situés dans le répertoire /lib/modules/$(uname -r)/kernel sous forme de fichiers ayant l'extension **.ko**.

Question 20 : *De la même manière que ci-dessus, déterminez le driver kernel utilisé pour gérer l'adaptateur branché à la question 18.*

Question 21 : *En analysant le fichier de configuration du kernel présent dans le /boot, déterminez s'il s'agit d'un driver fourni en module ou non. S'il s'agit d'un module, trouvez le fichier .ko associé.*

Les dépendances entre modules sont définies dans le fichier /lib/module/$(uname -r)/modules.dep.

Question 22 : *En regardant dans ce fichier, déterminez la/les dépendance(s) du driver utilisé pour gérer l'adaptateur précédemment connecté.*

La commande **lsmod** permet de lister les drivers modules actuellement chargés et indique le nombre de modules dépendants.

Question 23 : *Grâce à cette commande, vérifiez la réponse de la question 22.*

La commande **rmmod** permet de décharger un module.

Question 24 : *Que se passe t-il si vous essayez d'utiliser rmmod sur le module dont dépend le driver de l'adaptateur?*

Question 25 : *Utilisez rmmod sur le module driver de l'adaptateur et vérifiez l'effet dans le /dev.*

Question 26 : *Quelle est la différence entre la commande modprobe et la commande insmod?*

Question 27 : *Utilisez une de ces commandes pour recharger le module de l'adaptateur et vérifiez l'effet dans le /dev.*

4. Exercice 3 : /etc, udev et syslog

Contrairement à beaucoup de système UNIX, il n'y a pas de représentation du matériel réseau dans /dev sous Linux.

Par exemple, on peut récupérer les interfaces réseau grâce à la commande suivante :

```
> /sbin/ifconfig
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 48:5b:39:0b:10:82  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
```

```
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.12  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::226:5eff:fe02:27ac  prefixlen 64  scopeid 0x20<link>
        ether 00:26:5e:02:27:ac  txqueuelen 1000  (Ethernet)
        RX packets 40820  bytes 26045057 (24.8 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 39619  bytes 5521236 (5.2 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Il y a donc deux interfaces réseau : **wlan0** et **eth0**. Mais il n'y a aucun fichier /dev/eth0 ou /dev/wlan0. Cependant, c'est quand même **udev** qui gère ces interfaces.

Question 28 : *De la même manière que ci-dessus, déterminez les différentes interfaces réseau présentes sur votre machine ainsi que les addresses MAC associées.*

L'utilitaire **udev** utilise des fichiers de configuration permettant par exemple d'écrire des règles customisées de gestion de matériel.

Question 29 : *Que pouvez vous dire du contenu du fichier /etc/udev/rules.d/70-persistent-net.rules? Selon vous, à quel moment sont utilisées ces règles?*

Il existe un moniteur d'évènements fourni par udev et utilisable comme suit :

```
> udevadm monitor
```

Question 30 : *Lancez le moniteur puis branchez / débranchez l'adaptateur distribué ou bien une simple clé USB. Qu'observez vous?*

Question 31 : *Localisez un fichier **uevent** dans le répertoire /sys puis simulez un évènement d'ajout de matériel avec la commande **echo "add" > /sys/…./uevent**. Que dit le moniteur de udev?*

Le kernel envoi des messages dans l'espace utilisateur notamment par des sockets Netlink. Ce type de socket est utilisé pour établir une communication IPC entre le kernel et les processus de l'espace utilisateur. Pour ouvrir un tel socket en C :

```
#include <linux/types.h>
#include <linux/netlink.h>

...
...

struct sockaddr_nl nls;
memset(&nls,0,sizeof(struct sockaddr_nl));
nls.nl_family = AF_NETLINK;
nls.nl_pid = getpid();
nls.nl_groups = -1;

...
...

int fd = socket(PF_NETLINK, SOCK_DGRAM, NETLINK_KOBJECT_UEVENT);

...
...
```

Question 32 : *À partir de l'ébauche précédente permettant d'ouvrir un socket Netlink, écrivez un code permettant de lire les messages envoyés par le kernel à l'aide des fonctions* ***bind***, ***select*** *et* ***recv****.*

Question 33 : *Utilisez syslog pour logger dans /var/log et dans le terminal le path du device ajouté dans le /dev ainsi que le driver utilisé.*

Question 34 : *Testez votre code en connectant par exemple une clé USB. Observez le driver utilisé, les numéros majeur/mineur, etc…*

Question 35 : *Vérifiez dans le fichier correspondant de /var/log l'effet du syslog.*

Question 36 : *Créez un script dans /etc/init.d afin que votre code soit exécuté dès le boot de la machine.*

Question 37 : *Écrivez une règle udev permettant d'obtenir un fichier spécial /dev/ttyADAPTER plutôt que /dev/ttyUSB0. Vérifiez le fonctionnement grâce à votre code de récupération d'évènements uevent.*