

Cahier de TPs

Linux - Administration avancée

jour 2

François MICAUX - ACTILIS - 26.10

Table des matières

JOUR 2

Module 5 Maîtriser la production de packages RPM 3

Module 6 Stockage, Swap, Systèmes de fichiers 9

Module 7 Périphériques, modules et udev 16

Module 8 Le noyau : compilation et installation 21

Module 9 Le noyau : paramètres de boot et sysctl 26

Module 5

Maîtriser la production de packages RPM

TP : empaqueter GNU Hello, servir et signer un dépôt

Objectif et machine

Transformer l'archive de **GNU Hello** en **paquet RPM**, publier ce paquet dans un **dépôt** que `dnf` consulte en HTTP, puis **signer** paquets et métadonnées pour que le client refuse tout ce qui ne l'est pas.

La machine

La VM `vm3-ks` (ou toute VM AlmaLinux 10 reliée à Internet), avec un compte qui dispose de `sudo`. Tout se fait en SSH, en utilisateur ordinaire.

```
$ ssh admin@vm3-ks
```

> **Le logiciel**
>
> Ce TP reprend l'archive `hello-2.12.3.tar.gz` du TP de compilation, rangée dans `~/src`. Le compilateur et `make` (groupe `Development Tools`) doivent être installés.

Sans l'archive, la récupérer :

```
$ mkdir -p ~/src
curl -o ~/src/hello-2.12.3.tar.gz \
  https://mirrors.kernel.org/gnu/hello/hello-2.12.3.tar.gz
```

Étape 1 : Préparer l'environnement

```
$ sudo dnf install rpm-build rpmdevtools createrepo_c rpm-sign gnupg2
rpmdev-setuptree
ls ~/rpmbuild
```

Questions :

1. Quels sont les six répertoires créés par `rpmdev-setuptree` ?
2. Pourquoi construit-on un paquet sans être `root` ?

Étape 2 : Le fichier SPEC

Ranger l'archive dans `SOURCES/` :

```
$ cp ~/src/hello-2.12.3.tar.gz ~/rpmbuild/SOURCES/
```

Puis écrire `~/rpmbuild/SPECS/hello.spec` avec un éditeur :

```
Name:           hello
Version:        2.12.3
Release:        1%{?dist}
Summary:        Programme d'exemple de la distribution GNU
License:        GPL-3.0-or-later
URL:            https://www.gnu.org/software/hello/
Source0:        https://mirrors.kernel.org/gnu/hello/%{name}-%{version}.tar.gz
BuildRequires:  gcc, make

%description
GNU Hello affiche un message d'accueil. Il sert d'exemple
de logiciel construit avec les outils GNU.
```

Étape 3 : Le fichier SPEC (suite)

À la suite du fichier :

```
%prep
%autosetup

%build
%configure
%make_build

%install
%make_install
rm -f %{buildroot}%{_infodir}/dir
%find_lang %{name}

%check
make check

%files -f %{name}.lang
%license COPYING
%doc README NEWS
%{_bindir}/hello
%{_mandir}/man1/hello.1*
%{_infodir}/hello.info*

%changelog
* Tue Oct 06 2026 Prénom Nom <adresse@exemple.org> - 2.12.3-1
- Première version packagée
```

Questions :

1. Que remplacent `%configure`, `%make_build` et `%make_install` ?
2. Quelle commande donne la valeur de `%{_bindir}` ?
3. Quelle section décrit la liste des fichiers du paquet ?

> **À savoir**
>
> Trois lignes sortent du modèle de base. `rm -f ...dir` retire le fichier d'index `info` partagé, qui appartient à un autre paquet. `%find_lang` réunit les traductions du programme dans une liste, que `%files -f` ajoute. `%check` lance les tests du logiciel pendant la construction.

Étape 4 : Construire par étapes

Extraire seulement, regarder l'arbre, puis construire le RPM binaire et le RPM source :

```
$ cd ~/rpmbuild/SPECS
rpmbuild -bp hello.spec
ls ~/rpmbuild/BUILD
```

Questions :

1. Quelle étape produit l'arbre qui apparaît dans `BUILD/` ?

```
$ rpmbuild -ba hello.spec
ls ~/rpmbuild/RPMS/*/ ~/rpmbuild/SRPMS
```

Questions :

2. Quels fichiers `rpmbuild -ba` a-t-il produits, et dans quels répertoires ?
3. Que se passe-t-il si une dépendance de construction manque ?

> **À savoir**
>
> `hello-debuginfo` et `hello-debugsource` (symboles de débogage) sont produits automatiquement. Le dépôt n'en a pas besoin.

Étape 5 : Inspecter le paquet

```
$ cd ~/rpmbuild/RPMS/x86_64
rpm -qpl hello-2.12.3-1.el10.x86_64.rpm | grep -v locale
```

Questions :

1. Où le paquet installe-t-il le binaire et la page de manuel ?

```
$ rpm -qpi hello-2.12.3-1.el10.x86_64.rpm
```

Questions :

2. Que donne le champ `Signature` du paquet ?

```
$ rpm -qp --requires hello-2.12.3-1.el10.x86_64.rpm
```

Questions :

3. Le SPEC ne contient aucune ligne `Requires:`. D'où viennent les dépendances affichées ?

Étape 6 : Installer le paquet

```
$ sudo dnf install ./hello-2.12.3-1.el10.x86_64.rpm
rpm -qf /usr/bin/hello
rpm -ql hello | wc -l
```

Questions :

1. Quelle commande donne le paquet qui possède `/usr/bin/hello` ?

```
$ type -a hello
```

Questions :

2. Si le TP de compilation a été fait : pourquoi `hello` lance-t-il encore la version de `/opt/hello` ?

> **Les deux installations**
>
> `hello` compilé à la main (`/opt/hello`, inconnu de `rpm`) et le paquet (`/usr/bin`, suivi par `rpm`) coexistent. Seul le second se désinstalle avec `sudo dnf remove hello`.

Étape 7 : Créer le dépôt

Le dépôt ne reçoit que le paquet binaire :

```
$ sudo mkdir -p /srv/repo/myrepo/Packages
sudo chown -R $USER /srv/repo
cp ~/rpmbuild/RPMS/x86_64/hello-2.12.3-1.el10.x86_64.rpm \
  /srv/repo/myrepo/Packages/
createrepo_c /srv/repo/myrepo/
ls /srv/repo/myrepo/repodata
```

Questions :

1. Quel fichier de `repodata/` est l'index des autres ?
2. Quelle extension portent les fichiers de métadonnées produits ?
3. Quelle option de `createrepo_c` choisit la compression ?

Étape 8 : Servir le dépôt et l'utiliser

Un serveur HTTP minimal, lancé en arrière-plan, suffit pour le TP :

```
$ python3 -m http.server 8080 --directory /srv/repo &
```

Déclarer le dépôt côté client (pour l'instant sans signature), dans `/etc/yum.repos.d/myrepo.repo` créé avec un éditeur lancé en root (*`sudo`*) :

```
[myrepo]
name=Mon dépôt local
baseurl=http://localhost:8080/myrepo/
enabled=1
gpgcheck=0
```

```
$ sudo dnf remove hello
sudo dnf --disablerepo='*' --enablerepo=myrepo list available
sudo dnf --disablerepo='*' --enablerepo=myrepo install hello
```

Questions :

1. Quel est le nom du dépôt affiché par `dnf` pour la commande `list available` ?
2. Pourquoi ajouter `--disablerepo='*' --enablerepo=myrepo` ?

> **Un serveur de TP**
>
> `python3 -m http.server` sert tout le répertoire sans contrôle d'accès ni HTTPS. Il remplace ici un vrai serveur web, avec un dépôt en lecture seule servi en HTTPS.

Étape 9 : Créer la clé et signer le paquet

Créer une clé sans phrase secrète (pour le TP), l'exporter dans le dépôt et la déclarer à `rpmsign`. Le script de

génération est un fichier à créer avec un éditeur, `/tmp/cle.batch` :

```
%no-protection
Key-Type: RSA
Key-Length: 3072
Name-Real: ACTILIS TP
Name-Email: tp@actilis.net
Expire-Date: 1y
%commit
```

Puis générer la clé, l'exporter et la déclarer :

```
$ gpg --batch --gen-key /tmp/cle.batch
gpg --armor --export "ACTILIS TP" > \
  /srv/repo/myrepo/RPM-GPG-KEY-actilis-tp
echo '%_gpg_name ACTILIS TP' >> ~/.rpmmacros
```

Signer le paquet **dans le dépôt** et le vérifier :

```
$ rpmsign --addsign /srv/repo/myrepo/Packages/hello-2.12.3-1.el10.x86_64.rpm
sudo rpm --import /srv/repo/myrepo/RPM-GPG-KEY-actilis-tp
rpm -K /srv/repo/myrepo/Packages/hello-2.12.3-1.el10.x86_64.rpm
```

Étape 10 : Exiger les signatures et provoquer les refus

Côté client, passer le dépôt en `gpgcheck=1` et `repo_gpgcheck=1`, avec la clé publique :

```
[myrepo]
name=Mon dépôt local
baseurl=http://localhost:8080/myrepo/
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=http://localhost:8080/myrepo/RPM-GPG-KEY-actilis-tp
```

```
$ sudo dnf clean all
sudo dnf remove hello
sudo dnf --disablerepo='*' --enablerepo=myrepo install hello
```

Lire l'erreur, puis **signer les métadonnées** et recommencer. Le paquet est signé, mais les métadonnées du dépôt ne le sont pas encore (et `createrepo_c --update` les reconstruit à chaque ajout) :

```
$ createrepo_c --update /srv/repo/myrepo/
gpg --detach-sign --armor /srv/repo/myrepo/repodata/repomd.xml
sudo dnf clean all
sudo dnf --disablerepo='*' --enablerepo=myrepo install hello
```

Questions :

1. Que dit `dnf` au premier essai, avant la signature de `repomd.xml` ?
2. Quelle option du fichier `.repo` contrôle la signature des paquets, et laquelle celle des métadonnées ?
3. Quelle commande contrôle localement la signature d'un fichier RPM, comme à l'étape précédente ?

> **À savoir**
>
> Dès qu'un paquet est ajouté ou modifié, `createrepo_c` réécrit `repomd.xml` : l'ancien `repomd.xml.asc` ne correspond plus et `dnf` répond `Bad PGP signature`. Il faut signer de nouveau après chaque génération. Un paquet publié sans `rpmsign` est refusé avec `Package ... is not signed`.

Étape 11 : Publier une nouvelle version *(facultative)*

Passer le paquet en `2.12.3-2`, le reconstruire, le publier et mettre le client à niveau :

```
$ cd ~/rpmbuild/SPECS
rpmdev-bumpspec -c "Reconstruction du paquet" hello.spec
```

Questions :

1. Que modifie `rpmdev-bumpspec` dans le SPEC ?

```
$ rpmbuild -bb hello.spec
cp ~/rpmbuild/RPMS/x86_64/hello-2.12.3-2.el10.x86_64.rpm \
  /srv/repo/myrepo/Packages/
rpmsign --addsign /srv/repo/myrepo/Packages/hello-2.12.3-2.el10.x86_64.rpm
createrepo_c --update /srv/repo/myrepo/
gpg --batch --yes --detach-sign --armor \
  /srv/repo/myrepo/repodata/repomd.xml
sudo dnf clean expire-cache
sudo dnf --disablerepo='*' --enablerepo=myrepo upgrade
```

Questions :

2. Quelle commande confirme la version installée après la mise à niveau ?

> **Arrêter le serveur de TP**
>
> `kill %1` arrête le `python3 -m http.server` lancé en arrière-plan dans cette session.

Module 6

Stockage, Swap, Systèmes de fichiers

TP : partitionner, formater, monter, puis LVM, snapshot et RAID 1

Objectif et machine

Préparer **deux disques vierges** : partitions, systèmes de fichiers, montages permanents et swap, puis un **VG LVM** (extension à chaud, réduction, snapshot, déplacement de données) et, en dernier, un **RAID 1** logiciel.

La machine

Une VM AlmaLinux 10 avec un compte `sudo`, en SSH, et **deux disques VirtIO vierges de 10 Go**, `/dev/vdb` et `/dev/vdc`, en plus du disque système `vda`.

> ⚠ Matériel à ajouter avant ce TP
>
> C'est le premier TP du cahier qui demande des disques supplémentaires : **ajouter à la VM deux disques VirtIO de 10 Go** *(dans l'hyperviseur, VM éteinte ou à chaud)*. Le nettoyage de fin de TP les laisse vierges, les TP « noyau » et « optimisation » les réutilisent.

```
$ ssh admin@vm3-ks
lsblk
```

> ❗ Ne toucher que `vdb` et `vdc`
>
> Les commandes de ce TP écrasent des données. Le disque `vda` porte le système : il n'apparaît dans aucune commande d'écriture.

Étape 1 : Identifier les disques

```
$ cat /proc/partitions
lsblk
```

Questions :

1. Quel préfixe de nom portent les disques de cette VM, et quel pilote les sert ?

```
$ sudo fdisk -l /dev/vdb
```

Questions :

2. Quelle ligne de `fdisk -l` indique le type de table de partitions de `vda` ? Et celui de `vdb` ?

Étape 2 : Partitionner en GPT

`sfdisk` lit un script de partitionnement. Trois partitions sur `vdb` (système de fichiers, swap, LVM) : créer le fichier `vdb.sfdisk` dans le répertoire personnel avec un éditeur, et y écrire :

```
label: gpt
,2G,L
,1G,S
,,V
```

Une seule partition sur `vdc` (LVM), dans le fichier `vdc.sfdisk` :

```
label: gpt
,,V
```

Appliquer les deux scripts :

```
$ sudo sfdisk /dev/vdb < ~/vdb.sfdisk
sudo sfdisk /dev/vdc < ~/vdc.sfdisk
```

```
$ sudo partprobe /dev/vdb /dev/vdc
lsblk -o NAME,SIZE,PARTTYPENAME,PARTUUID /dev/vdb /dev/vdc
sudo parted /dev/vdb print
```

Questions :

1. Quel type porte chacune des trois partitions de `vdb` ?
2. Quelle commande demande au noyau de relire la table ?
3. Dans quel cas la relecture immédiate n'est-elle pas possible ?

Étape 3 : Formater et monter

```
$ sudo mkfs.ext4 -L data /dev/vdb1
sudo tune2fs -l /dev/vdb1 | grep -E \
  'volume name|UUID|Reserved block count|Block count'
sudo mkdir -p /data
sudo mount /dev/vdb1 /data
findmnt /data
df -h /data
sudo umount /data
```

Rendre le montage **permanent** avec l'UUID, puis le tester **sans redémarrer** :

```
$ sudo blkid -s UUID -o value /dev/vdb1
```

Ajouter une ligne à la fin de `/etc/fstab`, avec un éditeur lancé en root (`sudo`), en remplaçant l'UUID par celui relevé :

```
UUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX  /data  ext4  defaults,nofail  0 2
```

Puis :

```
$ sudo systemctl daemon-reload
sudo mount -a
findmnt /data
```

Questions :

1. Pourquoi `UUID=` plutôt que `/dev/vdb1` dans `fstab` ?
2. À quoi sert `nofail` ?
3. Que vaut la valeur `pass` de `/data`, et pourquoi ?

Étape 4 : Un démontage bloqué

Faire tenir `/data` par un processus, puis tenter de le démonter :

```
$ sudo chown $USER /data
(cd /data && exec sleep 600) &
sudo umount /data
sudo fuser -v /data
```

Libérer le point de montage, puis le remonter :

```
$ sudo fuser -km /data
sudo umount /data
sudo mount -a
```

Questions :

1. Quel message `umount` affiche-t-il ?
2. Quel code figure dans la colonne `ACCESS` du processus `sleep`, et que signifie-t-il ?
3. Que fait `fuser -km` ?

Étape 5 : Un swap

```
$ sudo mkswap /dev/vdb2
sudo swapon /dev/vdb2
swapon --show
free -h
```

Le rendre permanent, puis relire la configuration :

```
$ sudo blkid -s UUID -o value /dev/vdb2
```

Ajouter une ligne à la fin de `/etc/fstab`, avec un éditeur lancé en root (*`sudo`*), en remplaçant l'UUID par celui relevé :

```
≡ UUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX  none  swap  defaults,nofail  0 0
```

Puis :

```
$ sudo swapoff /dev/vdb2
sudo systemctl daemon-reload
sudo swapon -a
swapon --show
```

Questions :

1. Quelle commande prépare la partition, et laquelle l'active ?
2. Combien de zones de swap `swapon --show` affiche-t-il, avec quelles priorités ?

Étape 6 : Un groupe de volumes et deux volumes logiques

Les partitions `vdb3` et `vdc1` deviennent des **PV**, réunis dans un **VG** :

```
$ sudo pvcreate /dev/vdb3 /dev/vdc1
sudo vgcreate vg_tp /dev/vdb3 /dev/vdc1
sudo vgs vg_tp
sudo lvcreate -L 2G -n lv_www  vg_tp
sudo lvcreate -L 1G -n lv_logs vg_tp
```

Formater (XFS et ext4), monter et inscrire dans `fstab` :

```
$ sudo mkfs.xfs -q /dev/vg_tp/lv_www
sudo mkfs.ext4 -q -L logs /dev/vg_tp/lv_logs
sudo mkdir -p /data/www /data/logs
```

Ajouter les deux lignes suivantes à la fin de `/etc/fstab`, avec un éditeur lancé en root (*`sudo`*) :

```
≡ /dev/vg_tp/lv_www   /data/www   xfs   defaults,nofail  0  0
/dev/vg_tp/lv_logs  /data/logs  ext4  defaults,nofail  0  2
```

Puis :

```
$ sudo systemctl daemon-reload
sudo mount -a
df -h /data/www /data/logs
ls -l /dev/mapper | grep vg_tp
```

> **Disque déjà utilisé**
>
> Si le disque a servi à un TP précédent, `lvcreate` propose d'effacer l'ancienne signature (`Wipe it? [y/n]`) : répondre `y`.

Questions :

1. Quelle taille a le VG, et d'où vient-elle ?
2. Sous quels deux chemins retrouve-t-on le volume `lv_www` ?

Étape 7 : Étendre à chaud

Les deux volumes sont montés et **restent utilisables** pendant l'opération :

```
$ sudo lvextend -L +1G    --resizefs /dev/vg_tp/lv_www
sudo lvextend -L +512M  --resizefs /dev/vg_tp/lv_logs
df -h /data/www /data/logs
sudo lvs vg_tp
```

Questions :

1. Quel outil agrandit le système de fichiers XFS, et lequel l'ext4 ?
2. Quelle option de `lvextend` agrandit le système de fichiers dans la foulée ?

Étape 8 : Réduire un volume ext4

Un volume ext4 se **réduit** à froid. Une première tentative, volume monté, montre ce qui manque : répondre `n` à la question posée.

```
$ sudo lvreduce -L -500M --resizefs /dev/vg_tp/lv_logs
```

Démonter, puis réduire pour de bon et remonter :

```
$ sudo umount /data/logs
sudo lvreduce -y -L -500M --resizefs /dev/vg_tp/lv_logs
sudo mount /data/logs
df -h /data/logs
```

Questions :

1. Que demande `fsadm` quand le volume est monté ?
2. Dans quel ordre le système de fichiers et le volume sont-ils réduits ?

Étape 9 : Pas de réduction sur XFS

> **Pas de réduction sur `lv_www`**
>
> XFS ne se réduit pas. Sur un XFS agrandi à chaud (étape 7), `lvreduce --resizefs` a pourtant réduit le **volume** sans toucher au système de fichiers, qui dépasse alors son support et ne se monte plus (AlmaLinux 10.0, `lvm2` 2.03.28).

Étape 10 : Un snapshot

Un fichier, une photo du volume, puis une modification. Créer `/data/www/index.html` avec un éditeur lancé en root (*`sudo`*), contenant `version 1`, puis prendre la photo :

```
$ sudo lvcreate -s -L 500M -n snap_www /dev/vg_tp/lv_www
```

Modifier le fichier (`version 2`) avec l'éditeur, puis :

```
$ sudo lvs vg_tp
```

Lire la photo et le volume d'origine :

```
$ sudo mkdir -p /mnt/snap
sudo mount -o ro,nouuid /dev/vg_tp/snap_www /mnt/snap
```

Questions :

1. Pourquoi l'option `nouuid` au montage ?

```
$ cat /mnt/snap/index.html /data/www/index.html
sudo umount /mnt/snap
```

Questions :

2. Que contient `index.html` dans le snapshot, et dans le volume d'origine ?
3. Que devient un snapshot dont les 500 Mo réservés sont saturés ?

Étape 11 : Revenir en arrière

Fusionner le snapshot dans l'origine (retour à `version 1`) :

```
$ sudo lvconvert --merge /dev/vg_tp/snap_www
```

La fusion attend que l'origine soit **réactivée** :

```
$ sudo umount /data/www
sudo lvchange -an vg_tp/lv_www
sudo lvchange -ay vg_tp/lv_www
sudo mount /data/www
cat /data/www/index.html
sudo lvs vg_tp
```

Questions :

1. Que montrent `cat` et `lvs` après la réactivation ?

> **À savoir**
>
> Avec l'origine montée, `lvconvert --merge` répond `Delaying merge since origin is open`. La fusion a lieu à la prochaine activation du volume et se termine en arrière-plan : si `lvs` montre encore `snap_www`, attendre quelques secondes puis lancer `sudo lvchange --refresh vg_tp/lv_www`.

Étape 12 : Libérer un disque *(facultative)*

Avant de retirer le disque `vdb` du groupe, déplacer ce qu'il contient :

```
$ sudo lvs -o lv_name,devices vg_tp
sudo pvmove /dev/vdb3
sudo lvs -o lv_name,devices vg_tp
```

Questions :

1. Sur quel(s) PV se trouvent les volumes avant et après `pvmove` ?

```
$ sudo vgreduce vg_tp /dev/vdb3
sudo pvremove /dev/vdb3
sudo vgs vg_tp
```

Questions :

2. Dans quel ordre les commandes `pvmove`, `vgreduce` et `pvremove` s'enchaînent-elles ?

Étape 13 : Remettre les disques à zéro

Retirer d'abord les lignes `fstab` ajoutées (`/data`, le swap `vdb2` et les deux volumes), puis tout défaire **dans l'ordre inverse** de la création :

Éditer `/etc/fstab` avec un éditeur lancé en root (*`sudo`*) pour retirer ces lignes, puis :

```
$ sudo umount /data/www /data/logs /data
sudo swapoff /dev/vdb2
sudo lvremove -y vg_tp
sudo vgremove -y vg_tp
sudo pvremove -y /dev/vdc1
sudo wipefs -a /dev/vdb1 /dev/vdb2 /dev/vdb
sudo wipefs -a /dev/vdc1 /dev/vdc
sudo systemctl daemon-reload
lsblk /dev/vdb /dev/vdc
```

> ⚠ `fstab` avant les disques
>
> Une ligne de `fstab` sans `nofail` qui désigne un disque disparu bloque le démarrage de la VM. Les lignes de ce TP portent `nofail`, mais les retirer reste préférable.

> À savoir
>
> `wipefs -a` efface les signatures (système de fichiers, LVM, table de partitions). `rmdir /data/www /data/logs /data` retire les points de montage devenus inutiles.

Étape 14 : Un RAID 1 logiciel *(facultative)*

Une partition de 1 Go de type RAID sur chaque disque, assemblées en miroir :

```
$ sudo dnf install mdadm
for d in vdb vdc; do echo ',1G,R' | sudo sfdisk --label gpt /dev/$d; done
sudo mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/vdb1 \
  /dev/vdc1
cat /proc/mdstat
```

Formater, monter, écrire un fichier, puis **déclarer un disque en panne** :

```
$ sudo mkfs.xfs -q /dev/md0
sudo mkdir -p /mnt/raid && sudo mount /dev/md0 /mnt/raid
```

Créer `/mnt/raid/fichier` avec un éditeur lancé en root (*`sudo`*), contenant `ok`, puis :

```
$ sudo mdadm /dev/md0 --fail /dev/vdc1
sudo mdadm --detail /dev/md0 | grep -E 'State|Working|Failed'
cat /mnt/raid/fichier
```

Questions :

1. Que lit-on dans `/proc/mdstat` sur le niveau, les membres et l'état de `md0` ?
2. Combien de disques le RAID 1 tolère-t-il de perdre ? Que montre `mdadm --detail` après la panne simulée, et le fichier est-il lisible ?

À savoir

Dans `/proc/mdstat`, `[UU]` désigne deux membres actifs, `[U_]` un membre manquant ou en panne.

Défaire

`sudo umount /mnt/raid`, `sudo mdadm --stop /dev/md0`, `sudo mdadm --zero-superblock /dev/vdb1 /dev/vdc1`, puis `wipefs -a` sur les partitions et sur les deux disques.

Module 7

Périphériques, modules et udev

TP : lire le matériel, manipuler des modules, écrire une règle udev

Objectif et machine

Lire le **matériel** vu par le noyau, retrouver le **module** d'un périphérique, charger et configurer un module, créer un **fichier spécial** à la main, puis laisser **udev** le faire avec une règle écrite par soi.

La machine

Une VM AlmaLinux 10, avec un compte qui dispose de `sudo`. Tout se fait en SSH.

```
$ ssh admin@vm3-ks
```

> **Un module « sans matériel »**
>
> La VM n'a pas de périphérique rare à manipuler. Le module `dummy` (interfaces réseau factices) sert de cobaye à l'étape 3, et un fichier monté en disque (`loop`) sert de « clé USB » à l'étape 8.

Étape 1 : Détecter le matériel

`lspci` et `lsusb` viennent des paquets `pciutils` et `usbutils`, absents d'une installation minimale :

```
$ sudo dnf install pciutils usbutils
lspci
lspci -n
lspci -nn | grep -i ethernet
lsusb
```

Les noms viennent de la base `pci.ids` :

```
$ ls -l /usr/share/hwdata/
grep -m1 -A3 '^1af4' /usr/share/hwdata/pci.ids
```

Questions :

1. Quels sont l'emplacement, la classe, le constructeur et le périphérique du contrôleur Ethernet dans la sortie de `lspci -n` ?
2. Quel fichier `lspci` consulte-t-il pour afficher des noms, et que devient le contrôleur Ethernet sans lui ?
3. Quelle commande met à jour cette base ?

Étape 2 : Quel module pour quel périphérique ?

Le contrôleur SATA de la VM sert d'exemple. Relever son identifiant, puis chercher le module dans `modules.alias` :

```
$ lspci -nn | grep -i sata
grep -i 'pci:v00008086d00002922' /lib/modules/$(uname -r)/modules.alias
modinfo ahci | head -5
lsmod | grep ahci
```

Questions :

1. Quel module `modules.alias` associe-t-il à ce contrôleur ?
2. Où se trouve le fichier du module, et dans quel format ?
3. Ce module est-il chargé ?

> **À savoir**
>
> Certains pilotes ne figurent pas dans `modules.alias` parce qu'ils sont intégrés au noyau : c'est le cas du pilote `virtio-pci` des autres périphériques de cette VM. `lspci -k` affiche le pilote utilisé (`Kernel driver in use`).

Étape 3 : Charger et décharger

Le module `dummy` crée des interfaces réseau factices :

```
$ modinfo dummy | head -5
sudo modprobe dummy
lsmod | grep dummy
ip -br link | grep dummy
sudo modprobe -r dummy
```

Le même, avec un paramètre sur la ligne de commande :

```
$ sudo modprobe dummy numdummies=2
ip -br link | grep dummy
sudo modprobe -r dummy
```

Questions :

1. Quelle commande donne le chemin du fichier d'un module, ses dépendances et ses paramètres ?
2. `modprobe dummy` seul crée-t-il une interface ? Et avec `numdummies=2` ?
3. Quelle commande décharge le module ?

Étape 4 : La configuration dans `/etc/modprobe.d`

Rendre le paramètre permanent. Créer `/etc/modprobe.d/tp-dummy.conf` avec un éditeur lancé en root (*`sudo`*) :

```
≡ options dummy numdummies=2
```

Puis charger le module sans rien préciser :

```
$ sudo modprobe dummy
ip -br link | grep dummy
```

Questions :

1. Que fait le fichier `tp-dummy.conf` au chargement du module ?

```
$ sudo modprobe -c | grep 'options dummy'
sudo modprobe -r dummy
```

Questions :

2. Combien de lignes `options dummy` la configuration contient-elle, et laquelle l'emporte ?

> ⚠ L'ordre des fichiers
>
> Les fichiers sont lus dans l'ordre alphabétique de leur nom, tous répertoires confondus. Un fichier nommé `dummy.conf` passerait **avant** celui de `systemd.conf` et verrait sa valeur écrasée : aucune interface ne serait créée.

Étape 5 : `blacklist` et `install`

Remplacer le contenu de `/etc/modprobe.d/tp-dummy.conf` avec un éditeur lancé en root (*`sudo`*) par :

```
blacklist dummy
```

Puis :

```
$ sudo modprobe dummy
lsmod | grep dummy
sudo modprobe -r dummy
```

Questions :

1. Le module `dummy` est-il chargé malgré `blacklist` ?

Remplacer le contenu du même fichier par :

```
install dummy /bin/false
```

Puis :

```
$ sudo modprobe dummy
echo "code retour : $?"
lsmod | grep -c dummy
```

Questions :

2. Que répond `modprobe` avec la ligne `install` ?

Retirer ensuite le fichier :

```
$ sudo rm /etc/modprobe.d/tp-dummy.conf
```

Étape 6 : Les fichiers spéciaux

```
$ ls -l /dev/vda /dev/null /dev/tty0
```

Questions :

1. Quel est le type, le majeur et le mineur de `/dev/vda` et de `/dev/null` ?

```
$ head -12 /proc/devices
```

Questions :

2. Que lit-on dans `/proc/devices` ?

Créer son propre fichier spécial avec le couple de `/dev/null` :

```
$ sudo mknod -m 666 /dev/monnull c 1 3
ls -l /dev/monnull
echo test > /dev/monnull
echo "code retour : $?"
sudo rm /dev/monnull
```

Questions :

3. Que se passe-t-il quand on écrit dans `/dev/monnull` ?

Étape 7 : Observer udev

Suivre les évènements pendant qu'un module crée une interface. Ouvrir une **seconde session SSH** pour le moniteur :

```
$ sudo udevadm monitor --udev --subsystem-match=net
```

Dans la première session :

```
$ sudo modprobe dummy numdummies=1
sudo modprobe -r dummy
```

Les informations et attributs d'un disque :

```
$ udevadm info -q all -n vda | head -12
udevadm info -a -p /sys/block/vda | head -14
```

Questions :

1. Quels évènements `udevadm monitor` affiche-t-il pour `modprobe dummy` puis `modprobe -r dummy` ?
2. Quels liens `S:` udev a-t-il créés pour `vda` ?
3. Pour quoi servent les lignes `ATTR{...}` d'`udevadm info -a` ?

> **À savoir**
>
> Arrêter le moniteur avec Ctrl-C.

Étape 8 : Une règle `SYMLINK`

Un fichier de 50 Mo, formaté avec un **label** et présenté comme un disque, joue la clé USB :

```
$ cd /tmp
truncate -s 50M tpudev.img
mkfs.ext4 -q -L TPDISK tpudev.img
L=$(sudo losetup -f --show tpudev.img)
echo $L
sleep 3
udevadm info -q property -n $L | grep ID_FS_LABEL
ls -l /dev/disk/by-label/
```

La règle : un lien `/dev/mondisque` pour ce label, et une trace dans le journal.
Créer `/etc/udev/rules.d/99-tp.rules` avec un éditeur lancé en root (*`sudo`*) :

```
ENV{ID_FS_LABEL}=="TPDISK", SYMLINK+="mondisque"
ACTION=="change", ENV{ID_FS_LABEL}=="TPDISK", RUN+="/usr/bin/logger -t udev TP : %k"
```

Étape 9 : Tester la règle

Simuler, puis **rejouer l'évènement** du disque : il est traité quelques secondes plus tard.

```
$ sudo udevadm test /sys/class/block/$(basename $L) 2>&1 | grep mondisque
sudo udevadm trigger --action=change --name-match=$L
sleep 3
ls -l /dev/mondisque
sudo journalctl -t udev -n 3
```

Détacher le disque, vérifier que le lien disparaît, puis nettoyer :

```
$ sudo losetup -d $L
sleep 2
ls /dev/mondisque
sudo rm /etc/udev/rules.d/99-tp.rules /tmp/tpudev.img
```

Questions :

1. Quelle clé de la règle compare le label, et quelle clé crée le lien ?
2. Vers quel périphérique `/dev/mondisque` pointe-t-il, et que devient-il quand le disque est détaché ?
3. Quelle commande rejoue les évènements et laquelle teste les règles sans toucher au périphérique ?

> **À savoir**
>
> Un disque `loop` ne produit que des évènements `change`, d'où `ACTION=="change"` dans la deuxième règle. Pour une vraie clé USB, la règle de la slide « Autres exemples de règles » (`ACTION=="add"`) convient. Les règles sont relues sans `udevadm control --reload`, mais les évènements sont traités en arrière-plan : d'où les `sleep`.

Module 8

Le noyau : compilation et installation

TP : compiler un noyau réduit, l'installer, ranger les fichiers

Objectif et machine

Compiler un noyau **depuis les sources de kernel.org** avec une configuration **réduite** (seulement ce que la VM utilise), l'installer, constater ce que `make install` ajoute à `/boot`, puis **tout défaire**.

La machine

Une VM AlmaLinux 10 avec un compte `sudo`, **2 CPU ou plus**, et un **disque supplémentaire de 10 Go** (`/dev/vdb`, vierge : celui du TP « Stockage », laissé vierge par son nettoyage, ou un disque ajouté à la VM) pour `/usr/src`. Tout se fait en SSH.

```
$ ssh admin@vm3-ks
```

> **Secure Boot**
>
> Un noyau compilé n'est pas signé : si Secure Boot est actif, le firmware refuse de le démarrer. L'étape 1 relève l'état de la VM, l'étape 7 en tient compte.

> **Durée**
>
> La compilation seule demande une quinzaine de minutes avec 2 CPU. Elle se lance en arrière-plan et se suit dans un journal.

Étape 1 : Le noyau en place

```
$ uname -r
ls -l /boot
```

Questions :

1. Quels fichiers de `/boot` portent la version du noyau, et à quoi sert chacun ?

```
$ sudo grubby --default-kernel
grep -c '=m$' /boot/config-$(uname -r)
grep '^CONFIG_MODULES=' /boot/config-$(uname -r)
```

Questions :

2. Combien de modules le noyau de la distribution compile-t-il ?

```
$ [ -d /sys/firmware/efi ] && echo UEFI || echo BIOS
mokutil --sb-state
```

Questions :

3. La VM démarre-t-elle en UEFI ou en BIOS ? Secure Boot est-il actif ?

Étape 2 : Le disque et les paquets

Un système de fichiers pour `/usr/src` sur le disque supplémentaire :

```
$ echo ',,L' | sudo sfdisk --label gpt /dev/vdb
sudo mkfs.ext4 -q -L usrsrc /dev/vdb1
sudo mount /dev/vdb1 /usr/src
sudo chown $USER /usr/src
df -h /usr/src
```

Les paquets de la slide « Prérequis de compilation » :

```
$ sudo dnf group install "Development Tools"
sudo dnf install ncurses-devel openssl openssl-devel \
  elfutils-libelf-devel bc flex bison perl dwarves rsync
```

> ⚠ Sans la commande `openssl`
>
> La compilation s'arrête à l'étape `GENKEY` avec `openssl: command not found` si seul `openssl-devel` est installé.

Étape 3 : Les sources

```
$ cd /usr/src
curl -O https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.12.112.tar.xz
```

Contrôler la somme (relevée sur kernel.org le 2026-10-06). Créer le fichier `linux.sha256` dans `/usr/src` avec un éditeur lancé en root (*`sudo`*), contenant la somme, deux espaces, puis le nom de l'archive :

```
164dc9d1f6c93c61a15e1f071c48379b467f2b17c469cce7223471968208ed03  linux-6.12.112.tar.xz
```

Puis contrôler et extraire :

```
$ sha256sum -c linux.sha256
tar xJf linux-6.12.112.tar.xz
du -sh linux-6.12.112
```

Questions :

1. De quelle branche de kernel.org la version `6.12.x` fait-elle partie, et pourquoi la choisir pour un serveur ?
2. Quelle taille occupent les sources extraites ?

> À savoir
>
> Le numéro de la dernière 6.12 évolue : la page kernel.org donne la version courante, et le nom d'archive comme la somme sont à adapter.

Étape 4 : Configurer

Repartir de la configuration en place, la réduire aux modules chargés, puis suffixer la version pour ne pas

confondre ce noyau avec celui de la distribution :

```
$ cd /usr/src/linux-6.12.112
make mrproper
cp /boot/config-$(uname -r) .config
yes "" | make localmodconfig
scripts/config --set-str LOCALVERSION "-tp"
scripts/config --disable DEBUG_INFO_BTF --disable \
  DEBUG_INFO_DWARF_TOOLCHAIN_DEFAULT
make olddefconfig
```

Comparer :

```
$ grep -c '=m$' /boot/config-$(uname -r) .config
make -s kernelrelease
```

Questions :

1. Combien de modules la configuration contient-elle avant et après `localmodconfig` ?
2. Quelle version `make kernelrelease` annonce-t-il ?
3. Que produit `make mrproper`, et que remet-il à zéro ?

> **À savoir**
>
> `yes ""` accepte la valeur par défaut des options apparues depuis la version de la distribution. Les deux options `--disable` retirent les informations de débogage (`DEBUG_INFO_NONE`), qui alourdissent surtout la dernière étape de la compilation.

Étape 5 : Compiler

En arrière-plan, pour survivre à une coupure SSH :

```
$ cd /usr/src/linux-6.12.112
date +%T
nohup make -j$(nproc) > ~/build.log 2>&1 &
tail -f ~/build.log
```

Arrêter `tail` avec Ctrl-C. À la fin :

```
$ grep 'is ready' ~/build.log
date +%T
ls -lh arch/x86/boot/bzImage /boot/vmlinuz-$(uname -r)
ls /lib/modules
```

Questions :

1. Quel fichier est le noyau bootable, et de quelle taille par rapport à celui de la distribution ?
2. Les modules compilés sont-ils déjà dans `/lib/modules` ?

> **À savoir**
>
> Si `make` s'arrête, le message d'erreur est dans `~/build.log` : `grep -n -B3 'Error' ~/build.log`. Relancer `make` reprend où il s'était arrêté.

Étape 6 : Installer les modules, puis le noyau

Dans cet ordre, la cible `install` supposant les modules déjà installés :

```
$ cd /usr/src/linux-6.12.112
sudo make modules_install INSTALL_MOD_STRIP=1
du -sh /lib/modules/6.12.112-tp /lib/modules/$(uname -r)
sudo make install
```

Ce que `make install` a ajouté :

```
$ ls -l /boot | grep tp
sudo sh -c 'cat /boot/loader/entries/*6.12.112-tp.conf'
sudo grubby --default-kernel
sudo grubby --info=ALL | grep -E '^(index|title)'
```

Questions :

1. Quelle taille occupent les modules installés, et celle des modules de la distribution ? Que fait `INSTALL_MOD_STRIP=1` ?
2. Quels fichiers `make install` a-t-il créés dans `/boot` ?
3. Quel est maintenant le noyau par défaut ?

> **Noyau par défaut**
>
> Avec Secure Boot actif, un redémarrage sans autre précaution tomberait sur un noyau refusé. L'étape suivante remet l'ancien noyau par défaut.

Étape 7 : Démarrer ce noyau, ou pas

Secure Boot actif (réponse de l'étape 1, VM UEFI seulement) : ne pas redémarrer sur le nouveau noyau. Remettre celui de la distribution par défaut :

```
$ sudo grubby --set-default /boot/vmlinuz-$(uname -r)
sudo grubby --default-kernel
```

Secure Boot inactif : redémarrer sur le noyau compilé, puis lire la version et les modules chargés.

```
$ sudo reboot
```

```
$ uname -r
lsmod | wc -l
ls /boot/vmlinuz-*
sudo grubby --set-default /boot/vmlinuz-<version de la distribution>
```

Questions :

1. Que répond `uname -r` après le redémarrage sur le noyau compilé ?
2. Pourquoi remettre le noyau de la distribution par défaut avant de continuer ?

> **À savoir**
>
> La version de la distribution est celle notée à l'étape 1 (`uname -r` avant le redémarrage). Au menu GRUB, une touche choisit un autre noyau pour ce seul démarrage.

Étape 8 : Tout défaire

Revenir sur le noyau de la distribution s'il ne l'est pas, puis retirer le noyau de test, ses modules et les sources :

```
$ uname -r
sudo kernel-install remove 6.12.112-tp
sudo rm -r /lib/modules/6.12.112-tp
ls /boot /lib/modules
sudo grubby --default-kernel
rm ~/build.log
cd ~
sudo umount /usr/src
sudo wipefs -a /dev/vdb1 /dev/vdb
```

Questions :

1. Quelle commande retire le noyau, son initramfs et son entrée de menu ?
2. Que reste-t-il à supprimer à la main, et pourquoi ?

> ⚠ Ne pas retirer le noyau en cours
>
> `kernel-install remove` sur le noyau sur lequel la machine tourne laisse un système sans `/boot` cohérent. `uname -r` donne le noyau actif : il ne doit pas être celui qu'on retire.

Module 9

Le noyau : paramètres de boot et sysctl

TP : interroger le noyau, la ligne de boot, /proc et sysctl

Objectif et machine

Interroger le **noyau en place**, ajouter un **paramètre de boot**, puis lire, modifier et **rendre durables** des paramètres de `/proc/sys` avec `sysctl`.

La machine

Une VM AlmaLinux 10, avec un compte qui dispose de `sudo`. Tout se fait en SSH. Une étape demande un redémarrage : la session SSH est à rouvrir ensuite.

```
$ ssh admin@vm3-ks
```

> ⚠ Paramètres du noyau
>
> Un paramètre mal choisi change le comportement de toute la machine. Ce TP ne touche qu'à des valeurs sans risque et les remet à leur état initial à l'étape 8.

Étape 1 : Interroger le noyau

```
$ uname -r
uname -a
cat /proc/version
journalctl -k -b | head -3
```

Écrire un message dans le *Kernel Ring Buffer*, puis le retrouver :

```
$ echo "Salut du TP" | sudo tee /dev/kmsg
sudo dmesg | tail -2
journalctl -k -n 2 --no-pager
```

Questions :

1. Quelle version du noyau tourne ? Où se lit-elle en dehors de `uname` ?
2. Quelle commande donne les messages du noyau au-delà du *ring buffer* ?
3. Le message « Salut du TP » est-il visible dans `dmesg` et dans `journalctl -k` ?

Étape 2 : La ligne de commande du noyau

```
$ cat /proc/cmdline
cat /proc/sys/kernel/printk
```

Le premier chiffre de `kernel/printk` est le niveau de messages affichés sur la console. Ajouter le paramètre `loglevel=5` à tous les noyaux installés, puis vérifier :

```
$ sudo grubby --update-kernel=ALL --args="loglevel=5"
sudo grubby --info=DEFAULT | grep args
```

La ligne de `/proc/cmdline` n'a pas encore changé : elle décrit le démarrage **en cours**.

Questions :

1. Quels paramètres la ligne de commande contient-elle, et que fait `ro` ?
2. Le paramètre ajouté est-il visible dans `/proc/cmdline` tout de suite ? Pourquoi ?

Étape 3 : Redémarrer et vérifier

```
$ sudo systemctl reboot
```

Se reconnecter après une minute, puis :

```
$ cat /proc/cmdline
cat /proc/sys/kernel/printk
```

Questions :

1. Le paramètre est-il présent ? Quelle valeur a pris le premier chiffre de `kernel/printk` ?

Étape 4 : /proc

```
$ ls -l /proc/$$/cwd
ls -l /proc/meminfo
grep -E 'MemTotal|MemAvailable' /proc/meminfo
ls /proc | grep -c '^[0-9]'
```

Lire puis modifier un paramètre **directement** dans `/proc/sys` :

```
$ cat /proc/sys/net/ipv4/ip_forward
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
sysctl net.ipv4.ip_forward
```

Questions :

1. Quelle est la taille des fichiers de `/proc`, et que représentent les répertoires numériques ?
2. Le routage IPv4 est-il activé à la fin ? Résiste-t-il à un redémarrage ?

Étape 5 : Sysctl

Remettre le routage à `0`, essayer d'autres paramètres, et lister les classes :

```
$ sudo sysctl -w net.ipv4.ip_forward=0
sysctl vm.swappiness
sudo sysctl -w vm.swappiness=10
```

Questions :

1. Quelle était la valeur de `vm.swappiness`, et quel est son effet ?

```
$ sudo sysctl -w kernel.pid_max=131072
sudo sysctl -w kernel.pid_max=4194304
sudo sysctl -w kernel.osrelease=8.0
```

Questions :

2. Pourquoi `kernel.osrelease` est-il refusé, même avec `sudo` ?

```
$ sysctl -a 2>/dev/null | cut -d. -f1 | sort -u
sysctl -a 2>/dev/null | wc -l
```

Questions :

3. Quelles classes de paramètres existent, et laquelle est la plus peuplée ?

Étape 6 : Cache et surcharge mémoire

Vider les caches et mesurer :

```
$ free -m | head -2
sudo sh -c 'echo 3 > /proc/sys/vm/drop_caches'
free -m | head -2
```

Relever les limites d'allocation :

```
$ grep -E 'MemTotal|SwapTotal|CommitLimit|Committed_AS' /proc/meminfo
sysctl vm.overcommit_memory vm.overcommit_ratio
```

Questions :

1. Quelle colonne de `free` a diminué après `drop_caches` ?
2. Avec `vm.overcommit_memory=2` et `vm.overcommit_ratio=50`, quelle limite d'allocation le noyau applique-t-il ? La retrouver dans `CommitLimit`.

Étape 7 : Rendre durable

Un fichier par sujet, dans `/etc/sysctl.d`. Créer `/etc/sysctl.d/90-tp.conf` avec un éditeur lancé en root (*`sudo`*) :

```
# TP : routage IPv4 et swap réduit
net.ipv4.ip_forward = 1
vm.swappiness = 10
```

Puis :

```
$ sysctl net.ipv4.ip_forward
sudo sysctl --system | tail -4
sysctl net.ipv4.ip_forward vm.swappiness
```

Redémarrer pour vérifier la persistance :

```
$ sudo systemctl reboot
sysctl net.ipv4.ip_forward vm.swappiness
```

Questions :

1. Que valait `net.ipv4.ip_forward` juste après la création du fichier, avant `sysctl --system` ?
2. Les valeurs résistent-elles au redémarrage ?

Étape 8 : Remettre à l'état initial

```
$ sudo rm /etc/sysctl.d/90-tp.conf
sudo sysctl -w net.ipv4.ip_forward=0 vm.swappiness=60
sudo grubby --update-kernel=ALL --remove-args="loglevel=5"
sudo grubby --info=DEFAULT | grep args
```

> **À savoir**
>
> Si `tuned` est installé et actif, son profil (par exemple `virtual-guest`) pose aussi des valeurs comme `vm.swappiness`. Un fichier de `/etc/sysctl.d` n'est pas forcément la dernière couche appliquée : `sysctl vm.swappiness` donne la valeur réelle.