Joris Monnet

INF3 dlm-b

Rapport OS TP2 - Kernel :

Introduction :

Le but de ce travail pratique est de compiler un noyau linux et de manipuler son fichier de configuration afin de voir l'incidence que cela a sur le système (Question 9). Ici, nous utilisons une machine virtuelle. Ce TP a été réalisé dans l'ordre des questions du fichier de consignes et de ce fait, le rapport qui suit, suit le même ordre.

1. Installation d'une machine virtuelle :

J'ai fait le choix de reprendre ma première machine utilisée pour le TP1, c'est un Debian 10.5.0 en mode console, nommé Monnet-01.

2. Vérification de la version du noyau :

```
root@Monnet-01:~# uname -a
Linux Monnet-01 4.19.0-12-amd64 #1 SMP Debian 4.19.152-1 (2020-10-18) x86_64 GNU/Linux
```

On voit ici que le noyau actuel de ma machine est le 4.19.0.

3. Téléchargement du noyau le plus récent :

Le noyau le plus récent est le 5.10.2 du 22/12/20 :

mainline:	5.10	2020-12-13	[tarball] [pgp] [patch]
stable:	5.10.2	2020-12-21	[tarball] [pgp] [patch]
stable:	5.9.16 [EOL]	2020-12-21	[tarball] [pgp] [patch]

Je m'appuie sur ce fichier du site de kernel pour télécharger et vérifier la signature du kernel : https://www.kernel.org/signature.html

J'installe d'abord curl :

```
root@Monnet-01:~# apt install curl
Lecture des listes de paquets... Fait
```

Je télécharge ensuite le kernel sur la machine virtuelle :

```
root@Monnet-01:~# curl -OL https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.10.2.tar.xz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   162  100   162    0     0    473      0 --:--:-- --:--:-- --:--:--   472
100  111M  100  111M    0     0  1209k      0  0:01:34  0:01:34 --:--:-- 1018k
```

Puis la signature :

```
root@Monnet-01:~# curl -OL https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.10.2.tar.sign
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   162  100   162    0     0    381      0 --:--:-- --:--:-- --:--:--   381
100   989  100   989    0     0   1308      0 --:--:-- --:--:-- --:--:--  1308
```

Suite à une erreur de ma part la machine Monnet-01 a été corrompue en perdant son disque, j'ai donc recommencer sur la machine Monnet-02, et ai fait des snapshots de la machine virtuelle cette fois.

Je décompresse le kernel :

```
root@Monnet-02:~# unxz linux-5.10.2.tar.xz
unxz : linux-5.10.2.tar : Erreur d'écriture : Aucun espace disponible sur le périphérique
```

À la suite d'une discussion avec les collègues, je me rends compte que mon disque est mal monté et manque de place, je corrige et cette fois ça fonctionne, on a bien le .tar :

```
root@Monnet-02:~# ls -l
total 995360
-rw-r--r-- 1 root root 1019238400 déc.  22 22:35 linux-5.10.2.tar
-rw-r--r-- 1 root root        989 déc.  22 22:41 linux-5.10.2.tar.sign
```

Pour vérifier la signature, je télécharge gpg via apt install gnupg2.

Ensuite je vérifie la signature comme indiqué dans le document du site kernel.org :

```
root@Monnet-02:~# gpg2 --verify linux-5.10.2.tar.sign
gpg: les données signées sont supposées être dans « linux-5.10.2.tar »
gpg: Signature faite le lun. 21 déc. 2020 13:32:28 CET
gpg:                avec la clef RSA 647F28654894E3BD457199BE38DBBDC86092693E
gpg: Impossible de vérifier la signature : No public key
```

Comme spécifié dans le document je me retrouve confronté à une erreur classique de manque de clé publique, je fais donc :

```
root@Monnet-02:~# gpg2 --locate-keys torvalds@kernel.org gregkh@kernel.org
gpg: /root/.gnupg/trustdb.gpg : base de confiance créée
gpg: clef 38DBBDC86092693E : clef publique « Greg Kroah-Hartman <gregkh@kernel.org> » importée
gpg:       Quantité totale traitée : 1
gpg:                     importées : 1
gpg: clef 79BE3E4300411886 : clef publique « Linus Torvalds <torvalds@kernel.org> » importée
gpg:       Quantité totale traitée : 1
gpg:                     importées : 1
pub   rsa4096 2011-09-23 [SC]
      647F28654894E3BD457199BE38DBBDC86092693E
uid           [ inconnue] Greg Kroah-Hartman <gregkh@kernel.org>
sub   rsa4096 2011-09-23 [E]

pub   rsa2048 2011-09-20 [SC]
      ABAF11C65A2970B130ABE3C479BE3E4300411886
uid           [ inconnue] Linus Torvalds <torvalds@kernel.org>
sub   rsa2048 2011-09-20 [E]
```

Afin de récupérer les clés publiques de Linus Torvalds et Greg Kroan-Hartman.

Ensuite j'indique que cette dernière est bonne :

```
root@Monnet-02:~# gpg2 --tofu-policy good 647F28654894E3BD457199BE38DBBDC86092693E
gpg: Changing TOFU trust policy for binding <key: 647F28654894E3BD457199BE38DBBDC86092693E, user id:
 Greg Kroah-Hartman <gregkh@kernel.org>> from auto to good.
```

Ensuite je revérifie la signature du fichier, en utilisant tofu pour prendre en compte que la clé est bonne comme spécifié ci-dessus :

```
root@Monnet-02:~# gpg2 --trust-model tofu --verify linux-5.10.2.tar.sign
gpg: les données signées sont supposées être dans « linux-5.10.2.tar »
gpg: Signature faite le lun. 21 déc. 2020 13:32:28 CET
gpg:                avec la clef RSA 647F28654894E3BD457199BE38DBBDC86092693E
gpg: Bonne signature de « Greg Kroah-Hartman <gregkh@kernel.org> » [totale]
gpg: gregkh@kernel.org: Verified 1 signature in the past 4 minutes.  Encrypted
     0 messages.
```

C'est bon, la signature est vérifiée, c'est Greg Kroah-Hartman qui en est la source. On peut donc maintenant extraire.

4. Extraction :

J'utilise la commande tar -xvf. Après cela, je vérifie que le dossier est bien extrait :

```
root@Monnet-02:~# ls -l
total 995364
drwxrwxr-x 24 root root       4096 déc.  21 13:30 linux-5.10.2
-rw-r--r--  1 root root 1019238400 déc.  22 22:35 linux-5.10.2.tar
-rw-r--r--  1 root root        989 déc.  22 22:41 linux-5.10.2.tar.sign
```

On voit bien qu'un nouveau dossier a été créer. Je me place dedans et affiche la liste de son contenu, le dossier a bien été extrait :

```
root@Monnet-02:~# cd linux-5.10.2/
root@Monnet-02:~/linux-5.10.2# ls -l
total 840
```

5. Préparation d'un fichier de configuration :

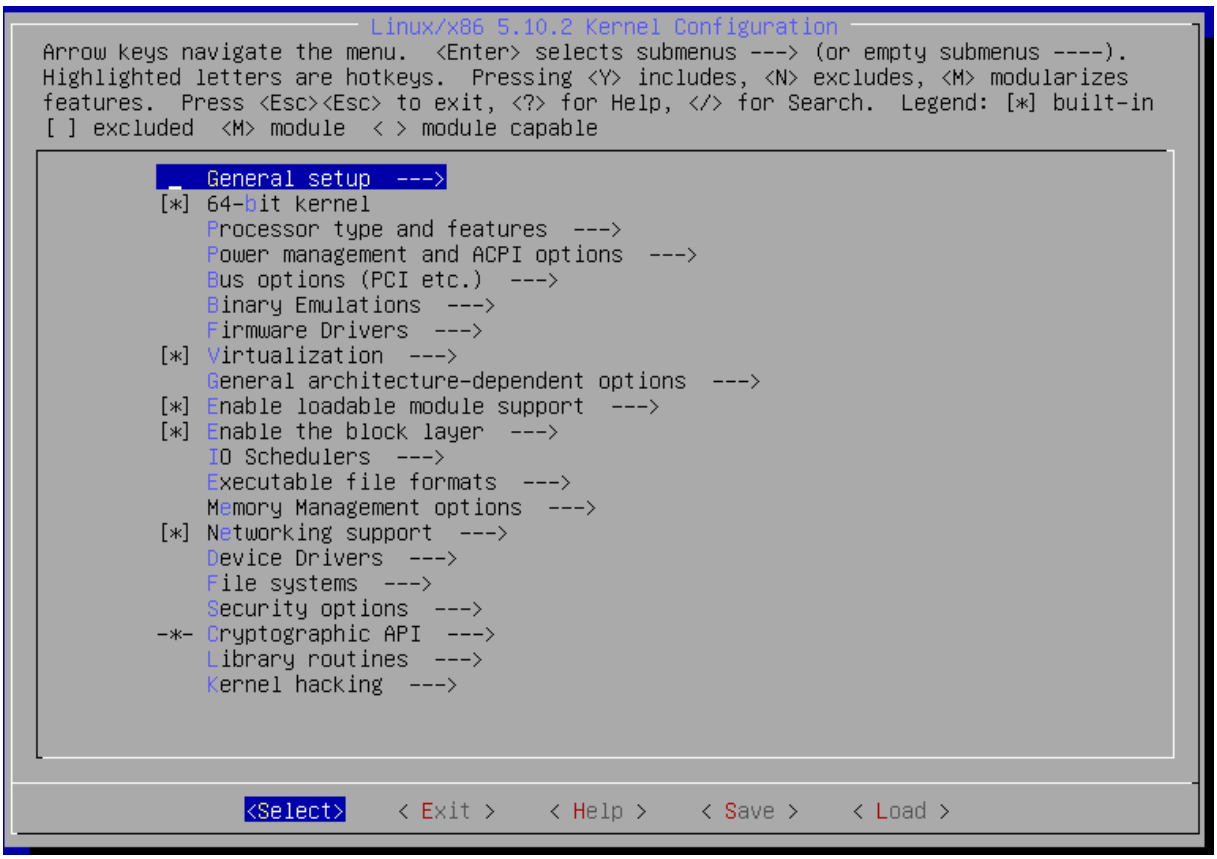
Je prépare un fichier .config : `root@Monnet-02:~/linux-5.10.2# touch .config`

Finalement cette solution ne convient pas, en effet partir d'un fichier de config vide complexifie la partie suivante utilisant make menuconfig, après divers essais, j'utilise donc finalement cette commande : `$ cp -v /boot/config-$(uname -r) .config afin de récupérer le fichier de config de mon ancien noyau.`

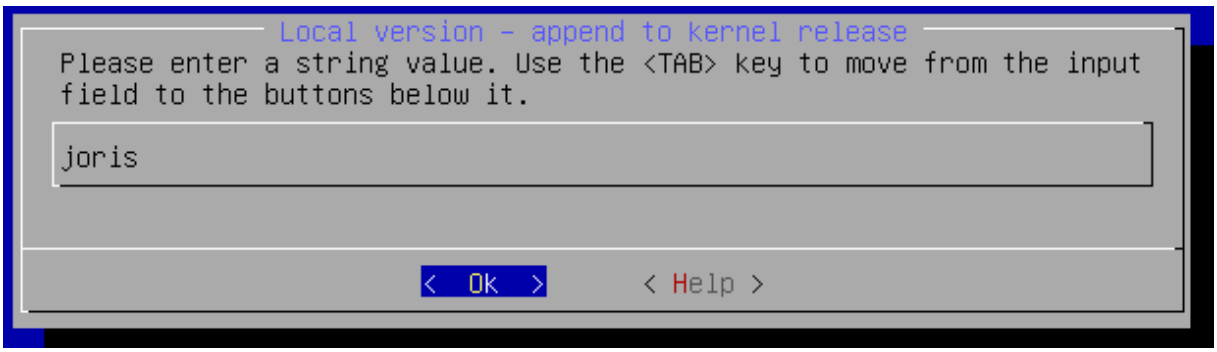
6. Modification du fichier de configuration :

J'installe make avec : `root@Monnet-02:~/linux-5.10.2/net# apt install make`

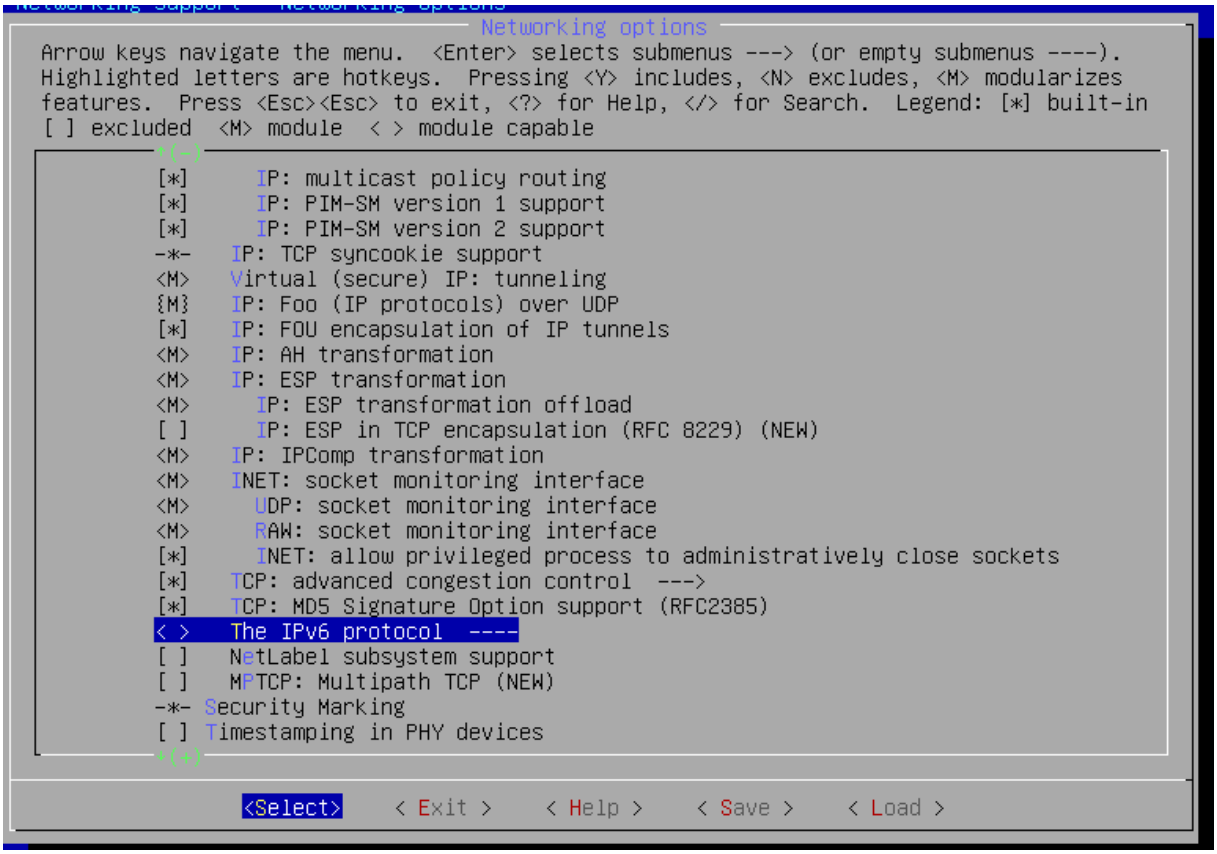
Ensuite j'installe gcc, pkg-config et libncurses-dev, bison, flex, utiles pour la commande make. Puis, je fais un make menuconfig pour commencer la configuration du kernel avec menuconfig, j'arrive sur cette page :



Pour la modification de uname je vais dans general settings -> Local version – append to kernel release :



Pour la suppression du protocole ipv6, je vais dans Networking, j'exclus le protocole IPV6 :



Je me suis aidé de ce site :
https://www.linuxtopia.org/online_books/linux_kernel/kernel_configuration/ch09s04.html

Je sauvegarde, puis fait un cat dans le .config, le fichier a bien été généré. On voit que les variables de configurations ont bien été ajoutées.

Après plusieurs essais avec différentes configurations qui ne marchaient pas (ne boot pas au démarrage), j'arrive enfin à une version viable à partir du fichier de config de l'ancien noyau.

7. Compilation et ajout du noyau au GRUB :

Maintenant que notre fichier de config est créé, on peut compiler avec make -j 5 pour utiliser plusieurs de mes cœurs de processeur (sans ça la commande make durait environ 4h30) :

```
error: Cannot generate ORC metadata for CONFIG_UNWINDER_ORC=y, please install libelf-dev, libelf-dev
el or elfutils-libelf-devel
make: *** [Makefile:1227: prepare-objtool] Error 1
```

Suite à cette erreur je télécharge les packages demandés ave apt install. Ensuite je recommence le make. Je m'aide de ce site https://www.cyberciti.biz/tips/compiling-linux-kernel-26.html .Une fois la commande make terminée, l'image est prête :

```
Kernel: arch/x86/boot/bzImage is ready  (#1)
```

Ensuite je fais make modules_install pour installer tous les modules nécessaires.

Ensuite, j'installe le kernel afin de mettre les fichiers nécessaires dans le répertoire /boot :

```
root@Monnet-02:~/linux-5.10.2# make install
sh ./arch/x86/boot/install.sh 5.10.2joris arch/x86/boot/bzImage \
        System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 5.10.2joris /boot/vmlinuz-5.10.2joris
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 5.10.2joris /boot/vmlinuz-5.10.2joris
update-initramfs: Generating /boot/initrd.img-5.10.2joris
find: '/var/tmp/mkinitramfs_tmIH4T/lib/modules/5.10.2joris/kernel': Aucun fichier ou dossier de ce t
ype
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 5.10.2joris /boot/vmlinuz-5.10.2joris
Création du fichier de configuration GRUB…
Image Linux trouvée : /boot/vmlinuz-5.10.2joris
Image mémoire initiale trouvée : /boot/initrd.img-5.10.2joris
Image Linux trouvée : /boot/vmlinuz-5.10.2joris.old
Image mémoire initiale trouvée : /boot/initrd.img-5.10.2joris
Image Linux trouvée : /boot/vmlinuz-4.19.0-13-amd64
Image mémoire initiale trouvée : /boot/initrd.img-4.19.0-13-amd64
Image Linux trouvée : /boot/vmlinuz-4.19.0-10-amd64
Image mémoire initiale trouvée : /boot/initrd.img-4.19.0-10-amd64
fait
```

Le make install ajoute tout seul le nouveau kernel a GRUB.

8. Redémarrage et vérification de version et d'IP :

Après avoir redémarré je fais un uname -a :

```
root@Monnet-02:~# uname -a
Linux Monnet-02 5.10.2joris #5 SMP Thu Dec 24 04:34:47 CET 2020 x86_64 GNU/Linux
```

On voit le résultat attendu, c'est-à-dire la version suivie de mon prénom et la date de compilation. Puis je vérifie la pile IP (en ayant au préalable installer net-tools) :

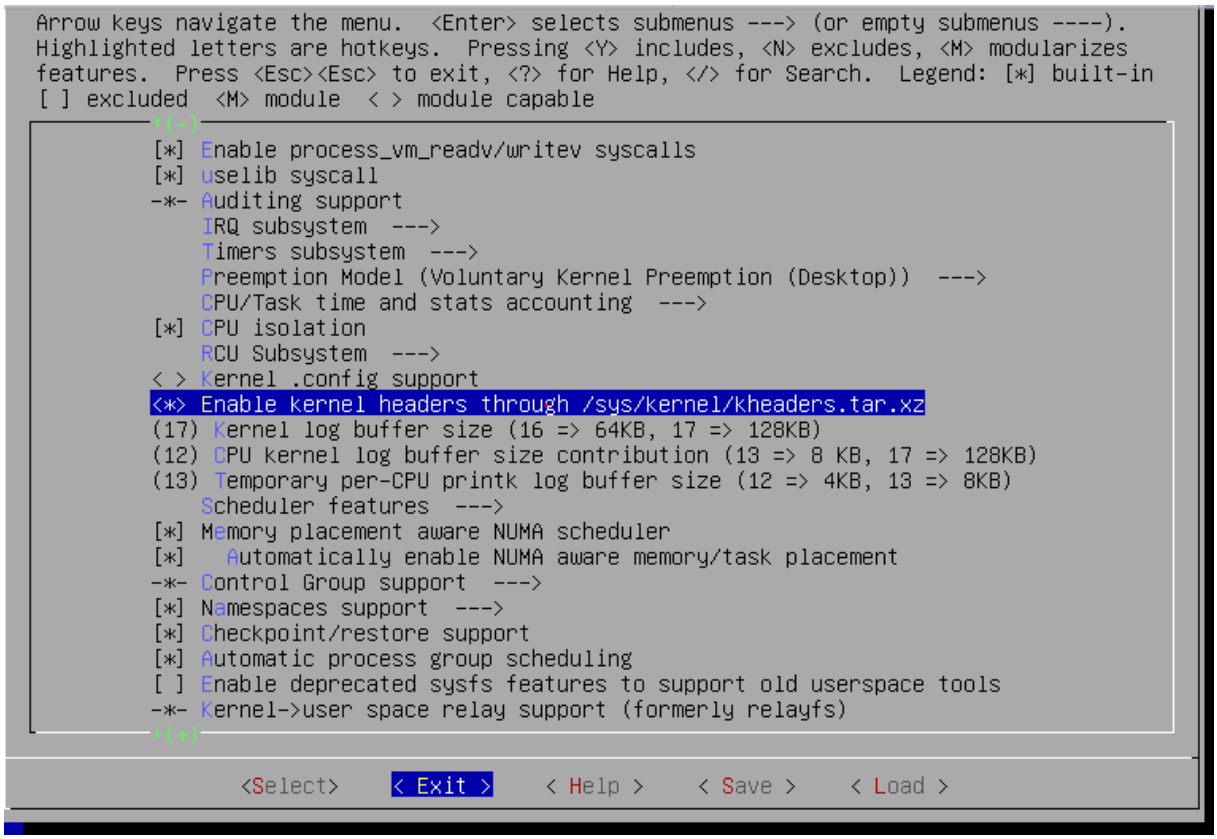
```
root@Monnet-02:~# ifconfig -a
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        ether 08:00:27:8c:f5:15  txqueuelen 1000  (Ethernet)
        RX packets 196  bytes 261201 (255.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 126  bytes 9178 (8.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Boucle locale)
        RX packets 1  bytes 29 (29.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1  bytes 29 (29.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

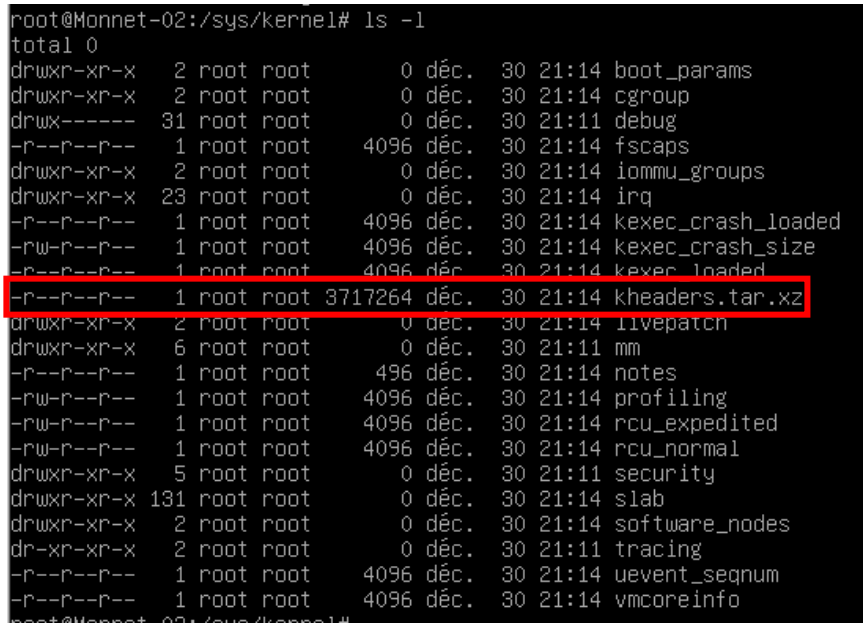
Il n'y a pas d'ipv6, c'est tout bon.

9. Modification du fichier de configuration :

Je décide pour commencer d'autoriser les headers de kernel dans /sys/kernel/kheaders.tar.xz :



Je refais, le make, le make modules_install, le make install et le reboot. Ensuite je fais un cd /sys/kernel puis ls -l :

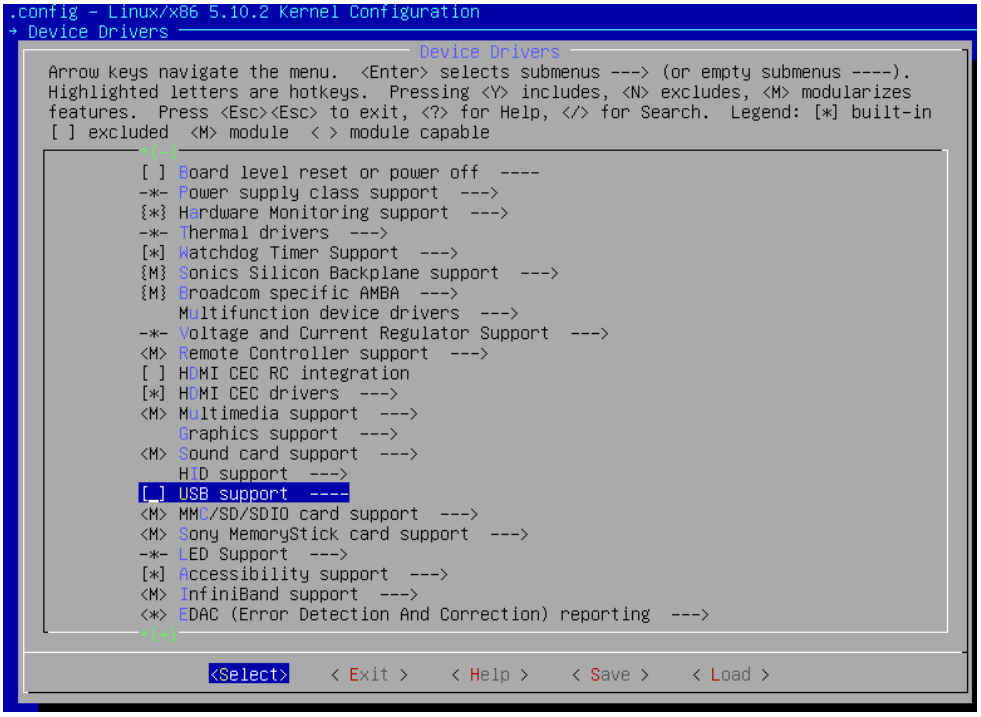


On trouve bien le fichier, qui n'est pas vide à l'endroit susmentionné. Afin de montrer la différence, j'affiche dans le même dossier la liste des fichiers avec mon ancienne version de kernel :

```
root@Monnet-02:/sys/kernel# ls -l
total 0
drwxr-xr-x   2 root root    0 déc.  30 21:20 boot_params
drwxr-xr-x   2 root root    0 déc.  30 21:20 cgroup
drwx------  29 root root    0 déc.  30 21:19 debug
-r--r--r--   1 root root 4096 déc.  30 21:20 fscaps
drwxr-xr-x   2 root root    0 déc.  30 21:20 iommu_groups
drwxr-xr-x  23 root root    0 déc.  30 21:20 irq
-r--r--r--   1 root root 4096 déc.  30 21:20 kexec_crash_loaded
-rw-r--r--   1 root root 4096 déc.  30 21:20 kexec_crash_size
-r--r--r--   1 root root 4096 déc.  30 21:20 kexec_loaded
drwxr-xr-x   2 root root    0 déc.  30 21:20 livepatch
drwxr-xr-x   6 root root    0 déc.  30 21:19 mm
-r--r--r--   1 root root  504 déc.  30 21:20 notes
-rw-r--r--   1 root root 4096 déc.  30 21:20 profiling
-rw-r--r--   1 root root 4096 déc.  30 21:20 rcu_expedited
-rw-r--r--   1 root root 4096 déc.  30 21:20 rcu_normal
drwxr-xr-x   4 root root    0 déc.  30 21:19 security
drwxr-xr-x 121 root root    0 déc.  30 21:19 slab
dr-xr-xr-x   2 root root    0 déc.  30 21:19 tracing
-r--r--r--   1 root root 4096 déc.  30 21:20 uevent_seqnum
-r--r--r--   1 root root 4096 déc.  30 21:20 vmcoreinfo
```

On voit que le fichier n'y était pas à la base.

Ensuite, je décide d'enlever l'accès à l'usb :



Je refais le make, make modules_install et make install. Ensuite je montre les usbs connectés :

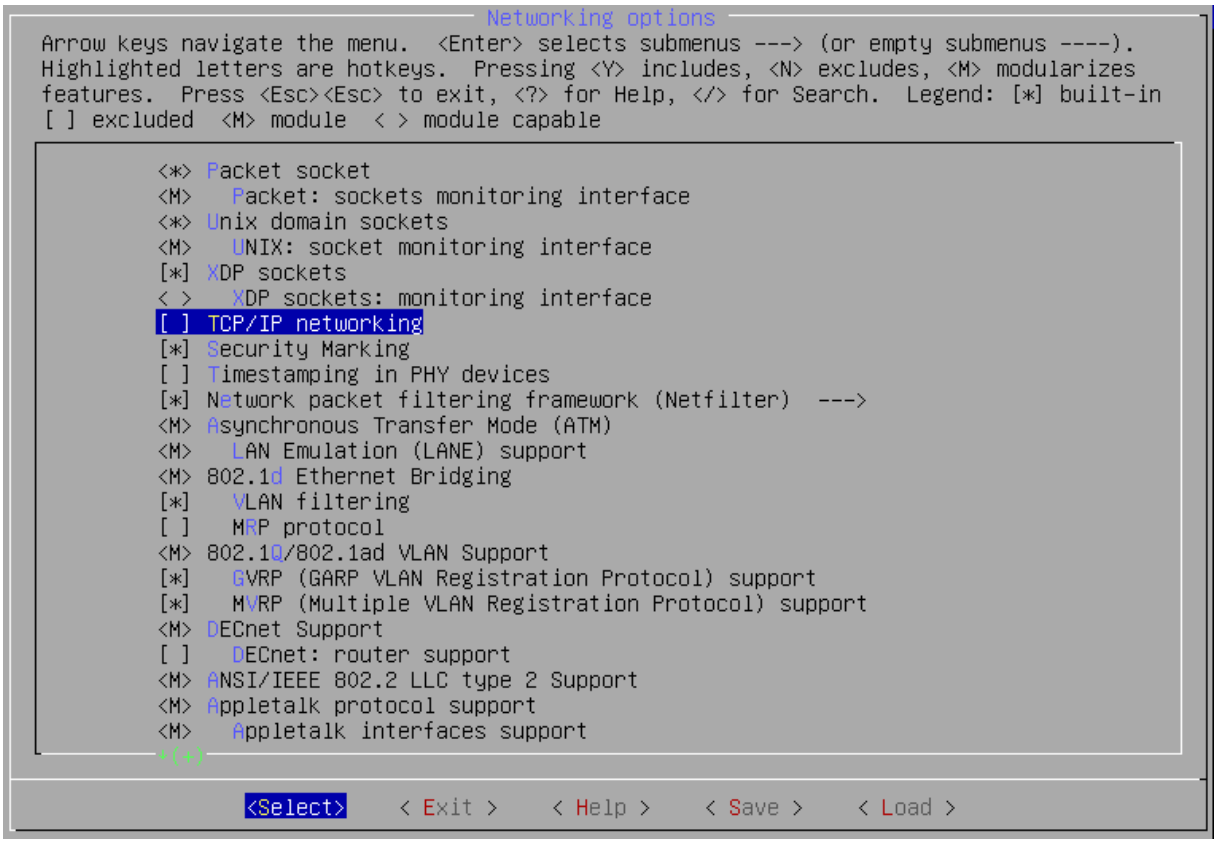
```
root@Monnet-02:~# lsusb
root@Monnet-02:~# _
```

Il n'y en a aucun comme prévu. Pour comparer je réalise la même commande dans mon ancien kernel :

```
root@Monnet-02:~# lsusb
Bus 001 Device 002: ID 80ee:0021 VirtualBox USB Tablet
Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
root@Monnet-02:~#
```

On voit ici que cette même commande nous montre tous les usbs connectés. Notre changement a donc bien fonctionné.

Finalement, je décide de retirer la possibilité de se connecter aux réseaux TCP/IP :



Je refais le make, make modules_install et make install, puis je reboot.

Une fois le kernel lancé, je fais un ifconfig -a :

```
root@Monnet-02:~# ifconfig -a
attention: pas de socket inet disponible: Aucun fichier ou dossier de ce type
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 08:00:27:8c:f5:15  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

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        loop  txqueuelen 1000  (Boucle locale)
        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
```

On voit qu'il n'y a pas de socket inet disponibles, pour rappel, ce sont ceux-ci qui donnaient les adresses ip dans la partie 8 :

```
root@Monnet-02:~# ifconfig -a
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        ether 08:00:27:8c:f5:15  txqueuelen 1000  (Ethernet)
        RX packets 196  bytes 261201 (255.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 126  bytes 9178 (8.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Boucle locale)
        RX packets 1  bytes 29 (29.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1  bytes 29 (29.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

L'accès a un réseau TCP/IP est donc bel et bien indisponible.

Conclusion :

Pour conclure, ce TP m'a permis de remettre en pratique mes connaissances dans les commandes linux utilisées au TP1. De plus, il m'est plus simple maintenant de comprendre comment fonctionne le noyau linux et de manière plus générale un système d'exploitation et ses connexions au hardware. La partie 9 m'aura notamment permis de tester les liens entre le kernel et certaines fonctions de bases de toute système d'exploitation comme l'USB ou le TCP/IP.