IV.1 – Configuration du réseau

Christophe BLAESS - janvier 2020

La configuration du système que nous avons réalisée jusqu'ici était relativement simple :

- ajout de *packages* choisis dans les *layers* disponibles avec Poky ou sur le site *Open Embedded Layer Index*,
- adaptations éventuelles des recettes avec l'aide de *patches* pour modifier certains fichiers,
- administration minimale du système produit (mots de passe, nom d'hôte).

Nous pouvons néanmoins avoir besoin de réaliser une configuration plus approfondie du système, c'est ce que nous évoquerons dans cette séquence.

Configuration de référence

Pour cette séquence, et les suivantes je vais repartir d'une image personnalisée en utilisant comme cible un Raspberry Pi 4. Je souhaite éviter l'émulation par QEmu pour disposer d'une vraie connexion réseau Ethernet. En outre le Raspberry Pi ne dispose pas de RTC (*Real Time Clock*) ce qui nous donnera l'occasion de gérer explicitement la mise à l'heure.

J'utilise donc un fichier `conf/local.conf` dans lequel les lignes suivantes ont été ajoutées

```
# <LOGILIN>

MACHINE="raspberrypi4"
DL_DIR="${TOPDIR}§/../downloads"
SSTATE_DIR="${TOPDIR}/../sstate-cache"
ENABLE_UART="1"

INHERIT += "extrausers"
EXTRA_USERS_PARAMS += "useradd  -p '\$5\$gFkqOeNU\$BOtEK2RhJIu
EXTRA_USERS_PARAMS += "usermod  -p '\$5\$IlhPmUHu\$AFPAiidalJ5
hostname_pn-base-files = "mybox"

# </LOGILIN>
 [...]
```

Le fichier `conf/bblayers.conf` intègre les *layers* suivants :

```
 [...]
BBLAYERS ?= " \
  /home/testing/Build/Lab/Yocto-lab/poky/meta \
  /home/testing/Build/Lab/Yocto-lab/poky/meta-poky \
  /home/testing/Build/Lab/Yocto-lab/poky/meta-yocto-bsp \
  /home/testing/Build/Lab/Yocto-lab/meta-raspberrypi \
  /home/testing/Build/Lab/Yocto-lab/meta-my-layer \
  /home/testing/Build/Lab/Yocto-lab/meta-openembedded/meta-oe
  "
```

Enfin, notre fichier `meta-my-layer/recipes-custom/images/my-image.bb` est configuré comme suit :

```
SUMMARY = "A customized image for development purposes."
LICENSE = "MIT"

inherit core-image

IMAGE_FEATURES += "splash"
IMAGE_FEATURES += "tools-debug"
IMAGE_FEATURES += "tools-profile"
IMAGE_FEATURES += "tools-sdk"
```

```
IMAGE_FEATURES += "ssh-server-dropbear"
IMAGE_FEATURES += "read-only-rootfs"
IMAGE_INSTALL_append = " mc"
IMAGE_INSTALL_append = " nano"
IMAGE_INSTALL_append = " my-scripts"
IMAGE_INSTALL_append = " python-modules"
IMAGE_INSTALL_append = " python-hello"
IMAGE_INSTALL_append = " hello-autotools"
IMAGE_INSTALL_append = " hello-cmake"
IMAGE_INSTALL_append = " hello-makefile"
IMAGE_INSTALL_append = " hello-simple"
```

J'ai relancé un *build* (qui a duré un bon moment car la dernière image pour Raspberry Pi datait de la séquence I.3).

```
[build-rpi]$ nice  bitbake  my-image
```

La commande «`nice`» précédant l'appel à `bitbake` demande à ce que la priorité des tâches lancées par ce dernier soit un peu moins élevée que la normale. Ainsi la compilation est un peu moins agressive vis-à-vis de l'ordonnancement et mon poste de travail reste plus confortable à l'utilisation pendant ce temps.

Notre image étant compilée et installée sur une carte micro-SD, nous pouvons démarrer le Raspberry Pi.

```
Poky (Yocto Project Reference Distro) 3.0.1 mybox ttyS0

mybox login: root
Password: (linux)
root@mybox:~# uname  -a
Linux mybox 4.19.75 #1 SMP Thu Dec 26 11:13:52 UTC 2019 armv7l
root@mybox:~# cat  /sys/firmware/devicetree/base/model
Raspberry Pi 4 Model B Rev 1.1
```

Configuration initiale du réseau

Au démarrage nous voyons ces traces qui ressemblent à des messages d'erreur :

```
udhcpc: started, v1.31.0
udhcpc: sending discover
udhcpc: sending discover
udhcpc: sending discover
udhcpc: no lease, forking to background
```

Par défaut, Yocto nous propose une image dont la configuration réseau est dynamiquement obtenue à l'aide du protocole DHCP.

Plus précisément, le Raspberry Pi envoie sur le réseau Ethernet un message *broadcast* disant «Mon adresse matérielle est XX:XX:XX:XX:XX, je souhaite obtenir une adresse IP». Normalement **un serveur DHCP** présent sur le réseau (généralement un routeur pour les réseaux d'enterprise ou une *box* dans un cadre domestique) lui répond en lui indiquant entre autres l'adresse IP qui lui est attribuée, l'adresse de la passerelle de sortie du sous-réseau et l'adresse du serveur DNS permettant de traduire les URL en adresse IP.

Mon Raspberry Pi n'était pas relié au réseau, nous voyons donc les messages d'échec de l'initialisation. Si je branche son port Ethernet sur le réseau local et que je le redémarre, les messages sont différents :

```
udhcpc: sending discover
udhcpc: sending select for 192.168.3.11
udhcpc: lease of 192.168.3.11 obtained, lease time 43200
```

Une fois connecté, je peux vérifier la configuration réseau.

```
Poky (Yocto Project Reference Distro) 3.0.1 mybox ttyS0

mybox login: root
Password: (linux)
root@mybox:~# ip  addr  show
1: lo:  mtu 65536 qdisc noqueue state UNKNOWN group default q]
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0:  mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether dc:a6:32:02:57:3d brd ff:ff:ff:ff:ff:ff
```

```
    inet 192.168.3.11/24 brd 192.168.3.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 2a01:e0a:22:ea90:dea6:32ff:fe02:573d/64 scope global
       valid_lft 86292sec preferred_lft 86292sec
    inet6 fe80::dea6:32ff:fe02:573d/64 scope link
       valid_lft forever preferred_lft forever
3: wlan0:  mtu 1500 qdisc noop state DOWN group default qlen 1
    link/ether dc:a6:32:02:57:3e brd ff:ff:ff:ff:ff:ff
root@mybox:~#
```

Il existe toutefois de nombreuses situations où on ne dispose pas de serveur DHCP sur notre réseau, et ceci pour différentes raisons (sécurité, politique d'administration, réseau industriel dédié à certains équipements…).

Nous pouvons modifier la configuration de Yocto pour que notre système dispose d'**une configuration réseau statique** avec des adresses fixées dès la production de l'image.

Pour cela, il nous faut configurer le fichier `/etc/network/interfaces` de la cible. Par défaut son contenu est le suivant

```
# The loopback interface
auto lo
iface lo inet loopback

# Wireless interfaces
iface wlan0 inet dhcp
        wireless_mode managed
        wireless_essid any
        wpa-driver wext
        wpa-conf /etc/wpa_supplicant.conf

iface atml0 inet dhcp

# Wired or wireless interfaces
auto eth0
iface eth0 inet dhcp
iface eth1 inet dhcp

# Ethernet/RNDIS gadget (g_ether)
# ... or on host side, usbnet and random hwaddr
iface usb0 inet static
```

```
        address 192.168.7.2
        netmask 255.255.255.0
        network 192.168.7.0
        gateway 192.168.7.1

# Bluetooth networking
iface bnep0 inet dhcp
```

Le moins que l'on puisse dire, c'est que la gamme des interfaces gérées par ce fichier est plutôt large :

- L'interface «`lo`» est initialisée dès le démarrage (mot-clé «`auto`») Il s'agit d'une interface IP (mot-clé «`inet`») fonctionnant en *loopback* (pour les communications internes au système).
- L'interface Wifi «`wlan0`» est configurée en poste client prêt à se connecter aux réseaux décrits dans le fichier `/etc/wpa_supplicant.conf`. Son adressage IP est obtenu dynamiquement (mot-clé «`dhcp`»).
- Les interfaces «`atml0`» (sans fil Atmel) et les deux interfaces Ethernet «`eth0`» et «`eth1`» sont configurées par le protocole DHCP.
- L'interface USB-net «`usb0`» est configurée statiquement : lorsqu'un hôte est relié par un câble USB à notre cible (qu'il voit comme un périphérique — *device* — USB), celle-ci initialise une communication réseau et prend l'adresse IP `192.168.7.2`. Cette configuration est normalement associée à la présence d'un serveur DHCP qui affecte à l'hôte l'adresse `192.168.7.1`.
- L'interface Bluetooth «`bnep0`» est également prête à recevoir une configuration dynamique par DHCP.

Ce fichier est très riche, mais en pratique son contenu utile est beaucoup plus réduit :

- le Raspberry Pi n'a pas d'interface «`atml0`», ni «`eth1`» ni port USB *device* permettant d'utiliser «`usb0`» (ses ports USB sont uniquement *hosts*).
- L'interface «`bnep0`» n'est pas utilisable sans le chargement de modules (et d'*overlays* pour le *device tree*) spécifiques.
- L'interface «`wlan0`» nécessite une configuration supplémentaire pour connaître les identifiants des réseaux auxquels se connecter.

Il nous reste au final deux interfaces utilisables immédiatement : «`lo`» et «`eth0`». Pour avoir une configuration statique de notre interface Ethernet, il suffirait donc que notre fichier `/etc/network/interfaces` ressemble à :

```
auto lo
```

```
iface lo inet loopback

auto eth0
iface eth0 inet static
  address  192.168.3.101
  netmask  255.255.255.0
  gateway 192.168.3.254
  dns-nameservers 8.8.8.8
```

La configuration du réseau Ethernet est évidement à adapter en fonction des situations. La ligne «`address`» configure l'adresse IP du Raspberry Pi, la ligne «`netmask`» le masque binaire du sous-réseau et celle «`gateway`» l'adresse de la passerelle à joindre pour accéder aux autres sous-réseaux et à Internet.

Une ligne est moins habituelle : «`dns-nameservers`» permet d'indiquer une liste de serveurs DNS disponibles, toutefois cette option ne fonctionne pas encore sur notre Raspberry Pi, car il manque un package dans notre image.

Configuration réseau statique

Si nous modifions manuellement le fichier `interfaces` de la cible et redémarrons, nous pouvons observer la configuration suivante :

```
Poky (Yocto Project Reference Distro) 3.0.1 mybox ttyS0

mybox login: root
Password: (linux)
root@mybox:~# ip  addr  show
1: lo:  mtu 65536 qdisc noqueue state UNKNOWN group default ql
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0:  mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether dc:a6:32:02:57:3d brd ff:ff:ff:ff:ff:ff
    inet 192.168.3.101/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 2a01:e0a:22:ea90:dea6:32ff:fe02:573d/64 scope globa
       valid_lft 86352sec preferred_lft 86352sec
```

```
    inet6 fe80::dea6:32ff:fe02:573d/64 scope link
       valid_lft forever preferred_lft forever
3: wlan0:  mtu 1500 qdisc noop state DOWN group default qlen 1
    link/ether dc:a6:32:02:57:3e brd ff:ff:ff:ff:ff:ff
root@mybox:~# ping  192.168.3.254
PING 192.168.3.254 (192.168.3.254): 56 data bytes
64 bytes from 192.168.3.254: seq=0 ttl=64 time=0.297 ms
64 bytes from 192.168.3.254: seq=1 ttl=64 time=0.248 ms
64 bytes from 192.168.3.254: seq=2 ttl=64 time=0.227 ms
^C
--- 192.168.3.254 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.227/0.257/0.297 ms
```

L'adresse IP de l'interface `eth0` est correcte, et on arrive à joindre une autre adresse du même sous-réseau (ici la passerelle). Essayons de joindre une adresse IP distante sur Internet :

```
root@mybox:~# ping  8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=57 time=9.055 ms
64 bytes from 8.8.8.8: seq=1 ttl=57 time=8.662 ms
64 bytes from 8.8.8.8: seq=2 ttl=57 time=8.785 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 8.662/8.834/9.055 ms
```

Ce serveur DNS de Google est bien accessible. Essayons de résoudre une adresse symbolique

```
root@mybox:~# ping  www.kernel.org
ping: bad address 'www.kernel.org'
root@mybox:~#
```

La résolution de noms échoue, car il nous manque le *package* «`resolvconf`» qui prend en compte la ligne «`dns-nameservers`» du fichier `interfaces`. Nous pouvons tout de suite rajouter la ligne suivante dans notre fichier d'image :

```
IMAGE_INSTALL_append = " resolvconf"
```

Toutefois, cela n'est pas suffisant. Nous avons modifié le fichier `interfaces` directement sur la cible, mais nous souhaitons qu'il soit configuré comme nous le voulons dès la production de l'image. Nous devons donc rechercher quelle recette le fournit.

Recherche de la recette responsable d'un fichier

Un problème revient régulièrement lorsque l'on cherche à affiner la configuration d'une image produite par Yocto : «mais quelle recette a bien pu installer ce fichier ?». Profitons de l'occasion offerte par le fichier «`interfaces`» pour voir comment procéder.

Nous allons employer **l'outil `devtool` fourni par Poky** pour qu'il nous indique quelle recette installe le fichier indiqué. La commande est exécutée sur la machine de développement, mais le chemin passé en argument correspond à celui observé dans l'arborescence de la cible :

```
[build-rpi]$ devtool  search  /etc/network/interfaces
NOTE: Starting bitbake server...
NOTE: Reconnecting to bitbake server...
NOTE: Retrying server connection (#1)...
NOTE: Reconnecting to bitbake server...
NOTE: Previous bitbake instance shutting down?, waiting to ret
NOTE: Retrying server connection (#2)...
Loading cache: 100% |#########################################
Loaded 2252 entries from dependency cache.
init-ifupdown         Basic TCP/IP networking init scripts and
```

Le nom du package est indiqué sur la dernière ligne : «`init-ifupdown`». Nous pouvons aussi demander à `devtool` de rechercher l'emplacement de la recette en question (même s'il est assez évident que c'est une recette de base de Poky) :

```
[build-rpi]$ devtool  find-recipe  init-ifupdown
NOTE: Starting bitbake server...
INFO: Creating workspace layer in /home/cpb/Yocto-lab/build-rp
NOTE: Reconnecting to bitbake server...
```

```
NOTE: Retrying server connection (#1)...
NOTE: Reconnecting to bitbake server...
NOTE: Previous bitbake instance shutting down?, waiting to ret
NOTE: Retrying server connection (#2)...
Parsing recipes: 100% |#####################################
Parsing of 1535 .bb files complete (0 cached, 1535 parsed). 22
/home/cpb/Yocto-lab/poky/meta/recipes-core/init-ifupdown/init-
```

Voyons le contenu du répertoire de cette recette :

```
[build-rpi]$ ls  ../poky/meta/recipes-core/init-ifupdown/
init-ifupdown-1.0  init-ifupdown_1.0.bb
[build-rpi]$ ls  ../poky/meta/recipes-core/init-ifupdown/init-
copyright  init  interfaces  nfsroot  qemuarm  qemuarm64  qemu
[build-rpi]$
```

Le fichier «`interfaces`» est bien là ! Il ne nous reste plus qu'à le remplacer en suivant la même méthode que celle que nous avions employée dans la séquence II.3. Commençons par créer les répertoires nécessaires dans notre *layer* :

```
[build-rpi]$ mkdir  -p  ../meta-my-layer/recipes-core/init-ifu
```

Puis nous y copions le fichier «`interfaces`» présenté plus-haut :

```
[build-rpi]$ nano  ../meta-my-layer/recipes-core/init-ifupdown
  auto lo
  iface lo inet loopback

  auto eth0
  iface eth0 inet static
    address  192.168.3.101
    netmask  255.255.255.0
    gateway 192.168.3.254
    dns-nameservers 8.8.8.8
```

Et enfin nous écrivons une petite extension `.bbappend` pour indiquer que les fichiers concernant cette recette doivent être recherchés prioritairement dans notre sous-répertoire :

```
[build-rpi]$ nano  ../meta-my-layer/recipes-core/init-ifupdown
  FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
```

Nous pouvons relancer le *build* et réinstaller notre image sur la carte SD du Raspberry Pi. Une fois le *boot* terminé, nous observons :

```
root@mybox:~# ip  addr  show
1: lo:  mtu 65536 qdisc noqueue state UNKNOWN group default ql
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0:  mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether dc:a6:32:02:57:3d brd ff:ff:ff:ff:ff:ff
    inet 192.168.3.101/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 2a01:e0a:22:ea90:dea6:32ff:fe02:573d/64 scope globa
       valid_lft 86384sec preferred_lft 86384sec
    inet6 fe80::dea6:32ff:fe02:573d/64 scope link
       valid_lft forever preferred_lft forever
3: wlan0:  mtu 1500 qdisc noop state DOWN group default qlen 1
    link/ether dc:a6:32:02:57:3e brd ff:ff:ff:ff:ff:ff
root@mybox:~# ping  www.kernel.org
PING www.kernel.org (136.144.49.103): 56 data bytes
64 bytes from 136.144.49.103: seq=0 ttl=53 time=17.712 ms
64 bytes from 136.144.49.103: seq=1 ttl=53 time=17.575 ms
^C
--- www.kernel.org ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 17.575/17.643/17.712 ms
```

La configuration est bien celle que nous avons fixée statiquement, la résolution de noms fonctionne ainsi que la communication avec une machine distante sur Internet.

Conclusion

Cette séquence nous a permis de personnaliser la configuration réseau, mais au-delà de cette action, nous avons vu comment expérimenter directement sur la cible

et reporter ensuite les modifications dans l'image en recherchant les recettes concernées et en les surchargeant.

«« **sommaire** »»