IV.2 – Configuration des utilitaires système

Christophe BLAESS - janvier 2020

Dans la séquence précédente, nous avons configuré le réseau à notre convenance. Toutefois, un problème persiste avec le Raspberry Pi…

Gestion de l'heure système

Mon Raspberry Pi a démarré depuis près d'une heure, comme l'indique la commande «`uptime`» :

```
root@mybox:/tmp# uptime
08:45:25 up 51 min,  1 user,  load average: 0.00, 0.00, 0.00
```

Je consulte l'heure du système avec la commande «`date`» :

```
root@mybox:/tmp# date
Fri Jan  3 08:46:40 UTC 2020
root@mybox:/tmp#
```

L'heure est correcte à quelques minutes près. Je décide alors de redémarrer mon Raspberry Pi :

```
root@mybox:/tmp# reboot

Broadcast message from root@mybox (ttyS0) (Fri Jan  3 08:47:41

The system is going down for reboot NOW!
```

Puis, une fois le redémarrage terminé, je consulte à nouveau l'heure système :

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

mybox login: root
Password: (linux)
root@mybox:~# date
Fri Jan  3 07:54:02 UTC 2020
root@mybox:~#
```

Surprise ! Nous avons voyagé dans le temps et sommes revenus près d'une heure en arrière…

Le Raspberry Pi n'a pas de composant RTC (*Real Time Clock*) pour maintenir l'heure lorsqu'il est éteint. Aussi au démarrage le système est-il à l'heure zéro d'Unix (aussi appelée l'«*epoch*») : le premier janvier 1970 à 00:00:00 UTC.

Outre le fait qu'un système évoluant en janvier 1970 paraît un peu ridicule à l'utilisateur, des problèmes peuvent se poser avec certains protocoles réseaux par exemple qui ne veulent pas accepter de certificats générés dans le lointain futur. Aussi, Yocto s'arrange-t-il pour que l'heure système soit initialisée avec une valeur réaliste : celle de production de l'image, qu'il a inscrite dans le fichier `/etc/timestamp` sous forme «année, mois, jour, heure, minute, seconde» juxtaposés :

```
root@mybox:~# cat  /etc/timestamp
20200103075346
```

L'inconvénient est qu'à chaque redémarrage le système reprend à la même heure, et qu'au fil du temps on s'éloigne de plus en plus de l'heure réelle. On préfèrerait une mise à l'heure automatique à partir du réseau. Pour cela il existe des serveurs NTP (*Network Time Protocol*) disponibles librement, il nous faut simplement un programme client `ntpd` pour les interroger.

```
root@mybox:~# ntpd
-sh: ntpd: command not found
```

Sur les systèmes embarqué, le client `ntpd` fait partie du *package* `busybox`, mais il n'a pas été intégré dans sa configuration. À nous de l'ajouter…

Configuration de Busybox

Busybox est un projet assez incontournable dans les systèmes restreints. Il est d'ailleurs surnommés «le couteau suisse de l'embarqué»…

Il s'agit d'un unique exécutable qui adapte son comportement en fonction du nom sous lequel il a été appelé. Cet exécutable implémente plus de 400 commandes Linux courantes (on choisit la liste des commandes à la configuration avant compilation), et des liens symboliques permettent de l'invoquer avec des noms différents. Lorsqu'on appelle `busybox` sans aucun argument il nous affiche la liste des commandes qu'il implémente :

```
root@mybox:~# busybox
BusyBox v1.31.0 (2019-12-26 11:35:27 UTC) multi-call binary.
BusyBox is copyrighted by many authors between 1998-2015.
Licensed under GPLv2. See source distribution for detailed
copyright notices.

Usage: busybox [function [arguments]...]
   or: busybox --list
   or: function [arguments]...

        BusyBox is a multi-call binary that combines many comm
        utilities into a single executable.  Most people will
        link to busybox for each function they wish to use and
        will act like whatever it was invoked as.

Currently defined functions:
        [, [[, addgroup, adduser, ash, awk, basename, blkid, b
        chmod, chown, chroot, chvt, clear, cmp, cp, cpio, cut,
        deluser, depmod, df, diff, dirname, dmesg, dnsdomainna
        env, expr, false, fbset, fdisk, fgrep, find, flock, fr
        grep, groups, gunzip, gzip, head, hexdump, hostname, h
        ifup, insmod, ip, kill, killall, klogd, less, ln, load
```

```
losetup, ls, lsmod, lzcat, md5sum, mesg, microcom, mkd
modprobe, more, mount, mountpoint, mv, nc, netstat, nd
pidof, pivot_root, printf, ps, pwd, rdate, readlink, r
rfkill, rm, rmdir, rmmod, route, run-parts, sed, seq,
shuf, sleep, sort, start-stop-daemon, stat, strings, s
sync, sysctl, syslogd, tail, tar, tee, telnet, test, t
udhcpc, udhcpd, umount, uname, uniq, unlink, unzip, up
which, who, whoami, xargs, xzcat, yes, zcat
```

Notre système est basé sur l'image «`core-image-base`» de Poky qui est assez complète, avec de nombreux *packages*, et le rôle de Busybox est relativement réduit. Si toutefois nous retournons sur l'image «`core-image-minimal`» que nous avons compilée dans la séquence I.3, nous voyons que Busybox est invoqué — par l'intermédiaire de liens symboliques — pour presque chaque appel de commande :

```
root@raspberrypi4:~# ls -l /bin/
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 ash -
lrwxrwxrwx    1 root     root            14 Dec 26 11:38 busyb
-rwxr-xr-x    1 root     root        476920 Dec 26 11:49 busyb
-rwsr-xr-x    1 root     root         42440 Dec 26 11:49 busyb
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 cat -
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 chatt
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 chgrp
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 chmod
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 chown
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 cp ->
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 cpio
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 date
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 dd ->
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 df ->
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 dmesg
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 dnsdo
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 dumpk
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 echo
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 egrep
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 false
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 fgrep
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 getop
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 grep
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 gunzi
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 gzip
```

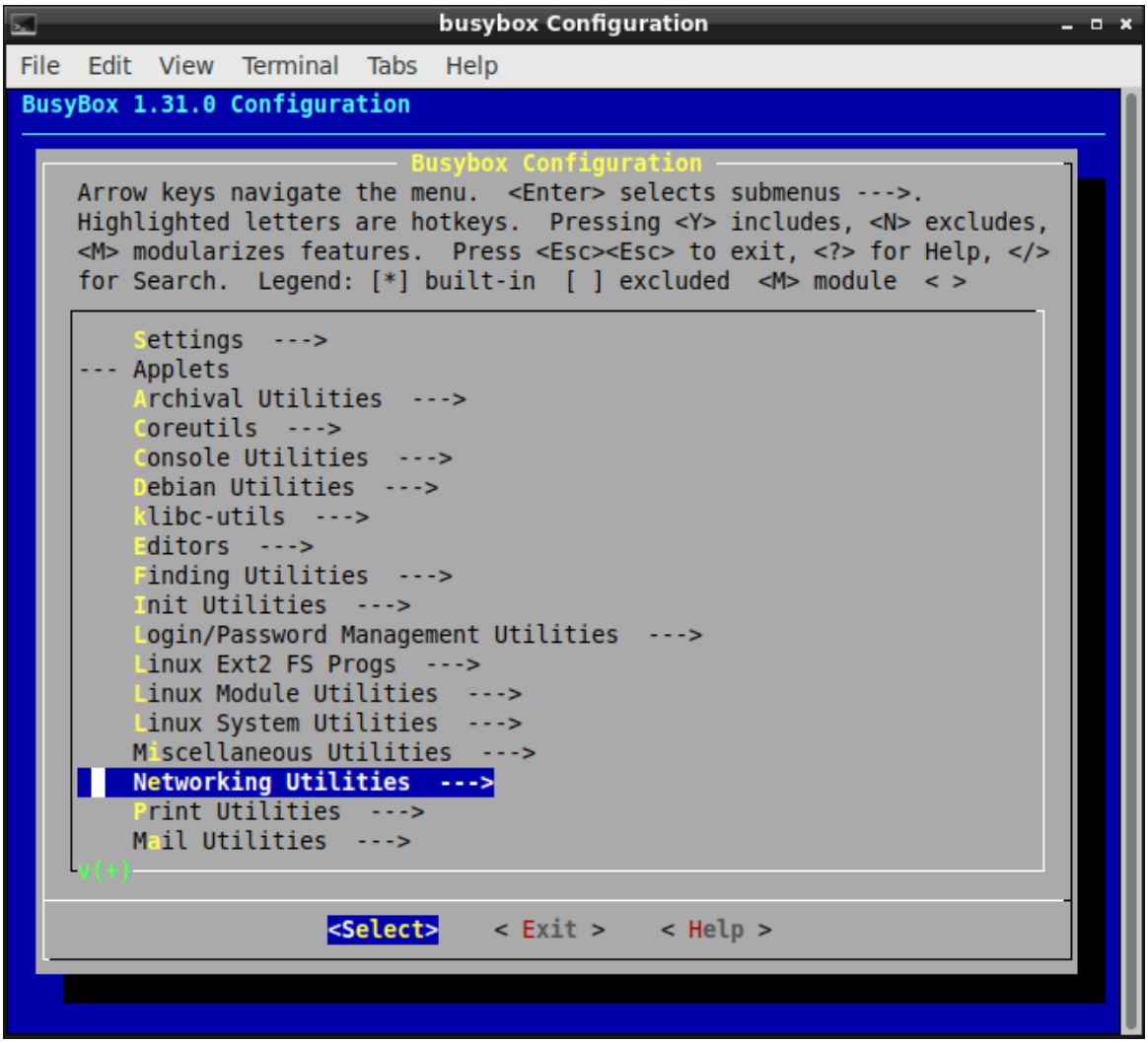
```
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 hostr
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 kill
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 ln ->
lrwxrwxrwx    1 root     root            17 Dec 26 12:49 logir
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 ls ->
lrwxrwxrwx    1 root     root            19 Dec 26 12:49 mkdir
```

Le suffixe «`.nosuid`» signifie simplement que c'est une commande ne nécessitant pas de privilège particulier pour son exécution.

Busybox nous propose une implémentation du client `ntpd` qui n'avait pas été sélectionnée dans la configuration par défaut. Pour accéder à la configuration, nous pouvons exécuter la commande suivante dans notre répertoire de *build* :

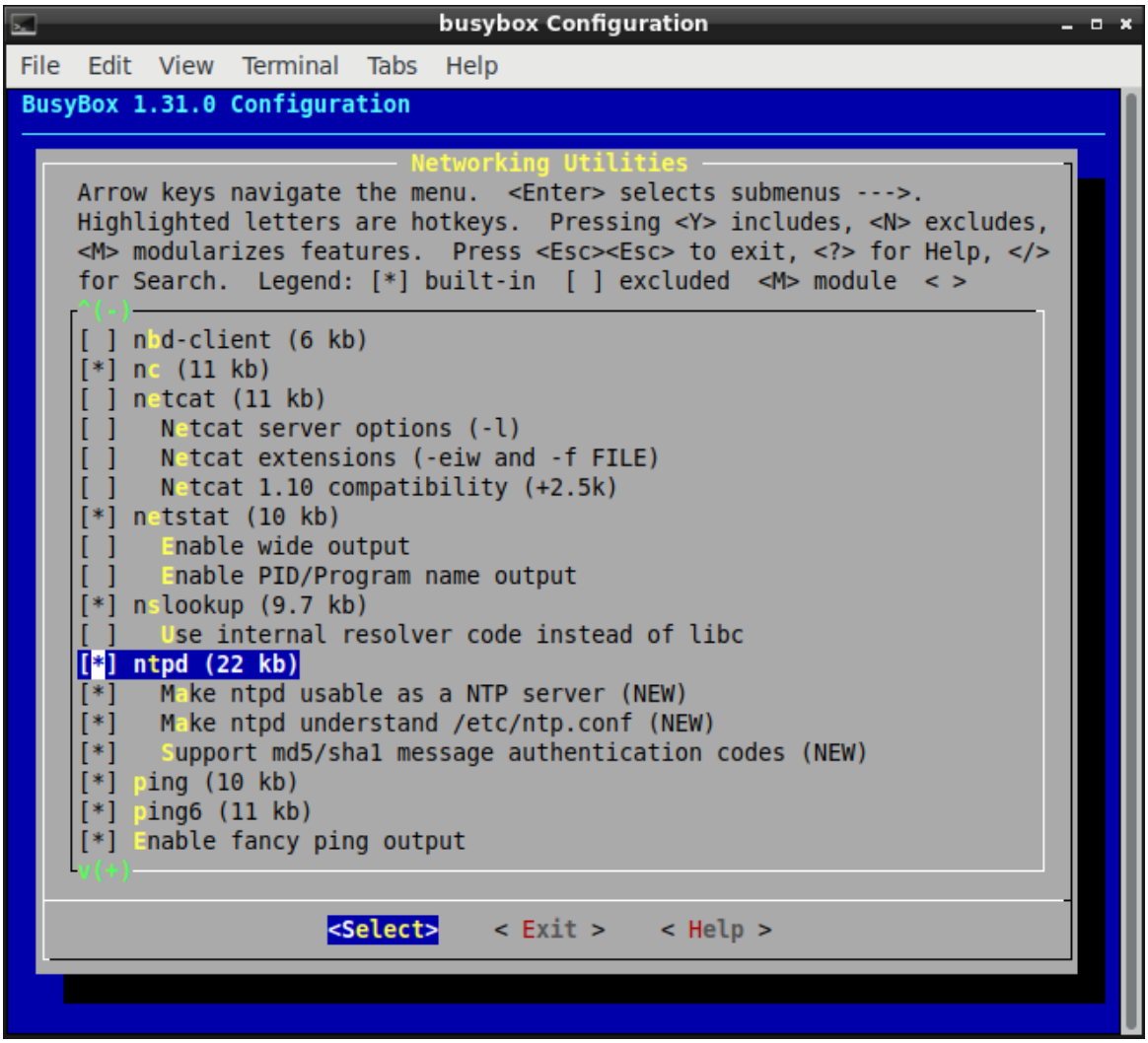
```
[build-rpi]$ bitbake  -c menuconfig  busybox
```

Cette commande demande à `bitbake` d'exécuter l'étape `menuconfig` de la recette correspondant à `busybox`. Il s'agit d'un menu de configuration proposé par le `Makefile` de Busybox. En principe une nouvelle fenêtre s'ouvre (ou un nouvel onglet de terminal) avec le menu de configuration de la figure IV.2-1

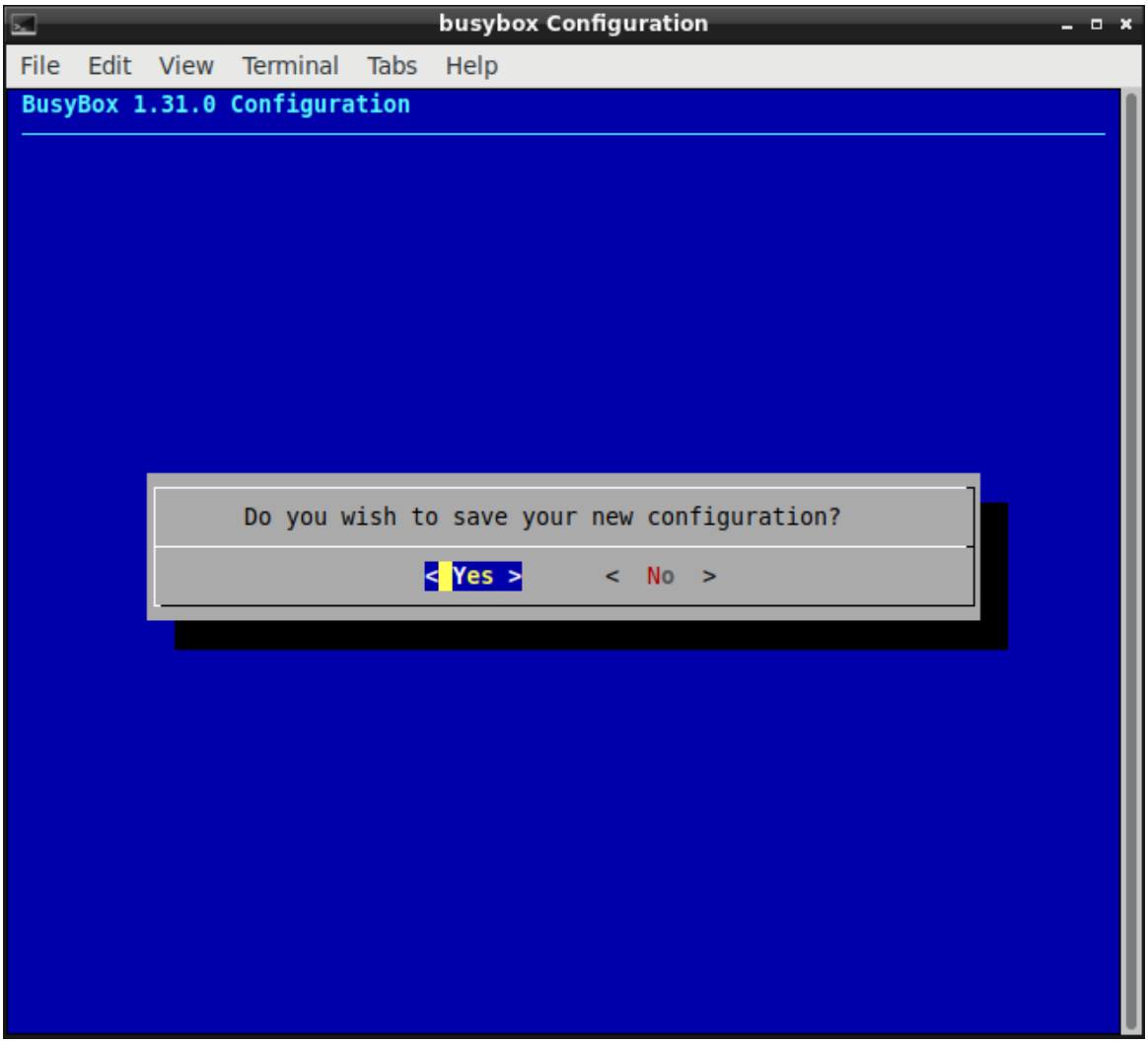


Dans certaines circonstances (connexion à distance, etc.) l'écran de configuration peut refuser de s'afficher. Dans ce cas, on peut contourner le problème en remplissant dans le fichier `conf/local.conf` la variable `OE_TERMINAL` avec la valeur «`screen`».

Nous accédons au menu «*Networking Utilities*» pour activer l'utilitaire `ntpd` comme sur la figure IV.2-2



Avant de quitter la configuration nous validons la sauvegarde de la configuration modifiée comme sur la figure IV.2-3



La configuration de Busybox a été modifiée uniquement dans le sous-répertoire que Yocto utiise pour son *build*. Mais nous souhaitons pérenniser cette modification dans notre image. Nous devons appeler `bitbake` pour réaliser une étape particulière de la recette :

```
[build-rpi]$ bitbake  -c  diffconfig  busybox
  [...]
Config fragment has been dumped into:
 /home/cpb/Yocto-lab/build-rpi/tmp/work/cortexa7t2hf-neon-vfp\
```

Cette étape *a priori* un peu obscure demande à `bitbake` de créer **un fragment de fichier de configuration** qui regroupe les modifications par rapport à la configuration par dédaut de la recette. Le fragment se trouve dans le chemin indiqué en fin de commande.

Nous devons intégrer ce fragment dans une extension de recette pour Busybox dans notre *layer*. Pour nous aider **l'outil `recipetool`** que nous avons déjà

rencontré lors de séquence II.3 va prendre le relais ainsi :

```
[build-rpi]$ recipetool  appendsrcfile  -w  ../meta-my-layer/
NOTE: Starting bitbake server...
Loading cache: 100% |#########################################
Loaded 2252 entries from dependency cache.
NOTE: Writing append file /home/cpb/Yocto-lab/meta-my-layer/re
NOTE: Copying tmp/work/cortexa7t2hf-neon-vfpv4-poky-linux-gnue
[build-rpi]$
```

On copie sur la ligne de commande de `recipetool` le chemin vers le fragment obtenu précédemment. Puis `recipetool` nous indique qu'il crée un fichier `.bbappend` et qu'il copie le fragment à l'endroit attendu par cette extension de recette.

Il nous reste à recompiler notre image. Nous effaçons la précédente compilation de Busybox d'abord pour éviter un *warning* de Yocto («*do_compile is tainted from a forced run*»).

```
[build-rpi]$ bitbake -c clean busybox
  [...]
[build-rpi]$ bitbake my-image
```

Après avoir recompilé et redémarré notre image, nous pouvons appeler `ntpd` implémenté par Busybox pour interroger le serveur de temps `pool.ntp.org`.

```
root@mybox:~# which  ntpd
/usr/sbin/ntpd
root@mybox:~# ls  -l  /usr/sbin/ntpd
lrwxrwxrwx 1 root root 19 Jan  3 12:51 /usr/sbin/ntpd -> /bin/
root@mybox:~# date
Fri Jan  3 12:54:38 UTC 2020
root@mybox:~# ntpd  -d  -n  -q  -p pool.ntp.org
ntpd: 'pool.ntp.org' is 51.15.182.163
ntpd: sending query to 51.15.182.163
ntpd: reply from 51.15.182.163: offset:+658.425551 delay:0.009
ntpd: sending query to 51.15.182.163
ntpd: reply from 51.15.182.163: offset:+658.425554 delay:0.009
ntpd: setting time to 2020-01-03 13:05:55.585718 (offset +658.
```

```
root@mybox:~# date
Fri Jan  3 13:05:58 UTC 2020
root@mybox:~#
```

Le système est maintenant à l'heure. Mais bien évidemment, au redémarrage…

```
root@mybox:~# reboot

Broadcast message from root@mybox (ttyS0) (Fri Jan  3 13:07:11
The system is going down for reboot NOW!
 [...]

Poky (Yocto Project Reference Distro) 3.0.1 mybox ttyS0

mybox login: root
Password: (linux)
root@mybox:~# date
Fri Jan  3 12:54:32 UTC 2020
root@mybox:~#
```

Il faudrait que la mise à l'heure soit automatiquement réalisée au démarrage du système. C'est ce que nous allons écrire à présent.

Démarrage d'un service

Il existe sous Linux plusieurs mécanismes pour lancer une tâche au démarrage. Les plus connues sont «`sysvinit`» («*System V Init*») et «`systemd`». De nos jours la plupart des PC et des serveurs utilisent «`systemd`». Les systèmes embarqués quant à eux utilisent encore majoritairement «*System V Init*».

Yocto choisit par défaut «*System V Init*» mais nous offre la possibilité d'utiliser «`systemd`» si on le préfère en configurant une recette de distribution spécifique. Ceci est un peu compliqué pour le cadre de ce cours en ligne, aussi allons-nous conserver l'environnement «`sysvinit`».

Le processus «`init`» lancé automatiquement par le noyau au *boot* consulte le fichier `/etc/inittab` pour savoir comment agir. Ce fichier configure le *runlevel* du système (son niveau de fonctionnement) à la valeur 5. Puis lance successivement le script *shell* `/etc/init.d/rcS` suivi de tous les scripts se trouvant dans le répertoire `/etc/rc5.d/` (le 5 correspondant au *runlevel*). Dans ce répertoire, on

trouve :

```
root@mybox:~# ls  /etc/rc5.d/
S01networking  S12rpcbind      S20bluetooth     S22ofono
S02dbus-1      S15mountnfs.sh  S20syslog        S64neard
S10dropbear    S20apmd         S21avahi-daemon  S99rmnologin.s
```

Les scripts présents commencent tous par la lettre «S» comme *Start* pour indiquer qu'ils sont lancés au démarrage du système, suivie d'un numéro d'ordre de démarrage. En réalité tous ces fichiers sont des liens symboliques vers des scripts se trouvant de `/etc/init.d` :

```
root@mybox:~# ls  -l  /etc/rc5.d/
total 0
lrwxrwxrwx 1 root root 20 Jan  3 12:51 S01networking -> ../ini
lrwxrwxrwx 1 root root 16 Jan  3 12:52 S02dbus-1 -> ../init.d/
lrwxrwxrwx 1 root root 18 Jan  3 12:52 S10dropbear -> ../init.
lrwxrwxrwx 1 root root 17 Jan  3 12:52 S12rpcbind -> ../init.d
lrwxrwxrwx 1 root root 21 Dec 26 09:01 S15mountnfs.sh -> ../in
lrwxrwxrwx 1 root root 14 Jan  3 12:51 S20apmd -> ../init.d/ap
lrwxrwxrwx 1 root root 19 Jan  3 12:52 S20bluetooth -> ../init
lrwxrwxrwx 1 root root 16 Jan  3 12:52 S20syslog -> ../init.d/
lrwxrwxrwx 1 root root 22 Jan  3 12:52 S21avahi-daemon -> ../i
lrwxrwxrwx 1 root root 15 Jan  3 12:52 S22ofono -> ../init.d/o
lrwxrwxrwx 1 root root 15 Jan  3 12:52 S64neard -> ../init.d/n
lrwxrwxrwx 1 root root 22 Dec 26 09:01 S99rmnologin.sh -> ../i
lrwxrwxrwx 1 root root 23 Dec 26 11:48 S99stop-bootlogd -> ../
```

Les liens symboliques sont créés au moment de la préparation du système de fichiers en prenant en compte les *runlevels* auxquels les services doivent démarrer et les dépendances entre ces services.

Nous pouvons créer un script qui lance automatiquement `ntpd` avec le nom du serveur NTP en argument. Pour être intégré correctement dans le système de démarrage, le script doit débuter avec un en-tête précis :

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:            ntpd
```

```
# Required-Start:       $networking
# Required-Stop:
# Default-Start:        2 3 4 5
# Default-Stop:         1
# Short-Description:    Network Time Protocol client Daemon
### END INIT INFO
#
```

Cet en-tête indique le nom du service fourni par ce script (lignes «*Provides*» et «*Short-Description*»), la liste des services dont il a besoin pour démarrer, ce qui permet de les ordonner. Nous demandons seulement la présence de «networking». Enfin on indique les *runlevels* dans lesquels le script doit être démarré et ceux où il doit être arrêté. Les listes «2 3 4 5» d'une part et «1» se retrouvent dans la plupart des *packages*. On consultera les scripts de `/etc/init.d/` pour avoir plus d'exemples.

C'est la classe «`update-rc.d`» fournie par Poky qui crée les liens en fonction des dépendances. Notre recette devra donc en hériter.

Le code permettant de lancer le service proprement dit vient à la suite de cet en-tête. On peut l'écrirer de différentes manières. La plus simple à mon avis est la suivante qui s'appuie sur la commande «`start-stop-daemon`» implémentée par Busybox :

```
do_start()
{
        start-stop-daemon --start --exec ntpd -- -p pool.ntp.o
}

do_stop()
{
        start-stop-daemon --stop --name ntpd
}


case "$1" in
  start) do_start ;;
  stop)  do_stop  ;;
  restart) do_stop; do_start ;;
  *) echo "Usage: $0 {start|stop|restart}" >&2; exit 1 ;;
esac
```

```
exit 0
```

Si notre script est appelé avec «`start`» en argument, il démarre le client NTP en enregistrant le numéro de processus (PID, *Process IDentifier*) dans un fichier. Par défaut `start-stop-daemon` le stocke dans `/var/run/<`*`nom-du-service`*`>.pid`. Si on appelle le script avec l'argument «`stop`», il arrête le client NTP en lui envoyant un signal (grâce au PID enregistré au démarrage).

Préparons un répertoire dans notre *layer* pour accueillir la recette et le script :

```
[build-rpi]$ mkdir  -p  ../meta-my-layer/recipes-custom/ntpd-s
```

Dans le sous-répertoire `files/` de ma recette, j'écris le script complet décrit ci-dessus.

```
[build-rpi]$ nano  ../meta-my-layer/recipes-custom/ntpd-start/
   [...]
```

Puis je rédige la recette suivante qui hérite de la classe `update-rc.d` et copie le script dans le répertoire `/etc/init.d/` de la cible en indiquant son nom dans la variable `INITSCRIPT_NAME` :

```
[build-rpi]$ nano  ../meta-my-layer/recipes-custom/ntpd-start/

SUMMARY = "Script to start ntpd client service"
SECTION = "custom"

LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835a

SRC_URI  = "file://ntpd"

inherit update-rc.d

INITSCRIPT_NAME = "ntpd"

do_install() {
  install -d ${D}${sysconfdir}
```

```
  install -d ${D}${sysconfdir}/init.d
  install -m 755 ${WORKDIR}/ntpd ${D}${sysconfdir}/init.d/ntpd
}
```

Après installation de l'image, le service est lancé dès le démarrage et le système est à l'heure :

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

mybox login: root
Password: (linux)
root@mybox:~# date
Sat Jan  4 09:52:18 UTC 2020
root@mybox:~# ls  /var/run/
agetty.reload  dropbear.pid  ld.so.cache  ntpd.pid     rpcbind
avahi-daemon   ifstate       lock         resolv.conf  rpcbind
dbus           klogd.pid     mount        resolvconf   syslogd
root@mybox:~# cat  /var/run/ntpd.pid
403
root@mybox:~# ps  aux
  [...]
root       403  0.0  0.0   2636  1432 ?        Ss   09:52   0:
  [...]
root@mybox:~# ls -l /etc/rc5.d/
total 0
  [...]
lrwxrwxrwx 1 root root 19 Jan  4 09:41 S20bluetooth -> ../init
lrwxrwxrwx 1 root root 14 Jan  4 09:42 S20ntpd -> ../init.d/nt
lrwxrwxrwx 1 root root 16 Jan  4 09:41 S20syslog -> ../init.d/
  [...]
```

Conclusion

Cette séquence nous a permis de voir comment modifier la configuration de Busybox et lancer un service au démarrage par l'intermédiaire d'un script de lancement.

Dans la séquence suivante, nous allons continuer l'adaptation de notre image pour une cible personnalisée, en s'intéressant cette fois aux interactions avec le matériel.