II.1 – Personnalisation de la configuration

Christophe BLAESS - janvier 2020

Dans la première partie, nous avons découvert le fonctionnement de base de Yocto. Nous avons réussi à produire et tester des images pour différentes plateformes. Toutefois nous n'avons absolument rien modifié au contenu de l'image. Dans cette nouvelle partie nous allons personnaliser notre système embarqué. Pour la plupart des constructions dans les séquences à venir nous emploierons une **cible Arm émulée par Qemu** pour la simplicité des tests et des captures d'écran. Toutefois les opérations décrites ici seront adaptables sur n'importe quelle carte supportée par Yocto.

Débutons donc cette séquence avec la commande :

```
[Yocto-lab]$ source  poky/oe-init-build-env  build-qemu
 [...]
[build-qemu]$
```

Jusqu'à présent nous n'avons modifié aucun fichier fourni par Yocto. Nous nous sommes contentés d'indiquer sur la ligne de commande la machine cible désirée par l'intermédiaire d'une variable d'environnement.

On se doute bien que cela n'est pas une solution pérenne. Tout d'abord on risque

d’oublier d'initialiser une variable au moment de lancer un *build*, mais en outre nous allons très vite avoir besoin de remplir une — voire plusieurs — dizaines de variables pour ajuster le contenu de notre système, ce qui n’est pas manipulable depuis la ligne de commande.

Fichier `local.conf`

Nous allons commencer par intervenir dans **le fichier «`local.conf`»** se trouvant dans le sous-répertoire «`conf/`» de notre répertoire de compilation. Ultérieurement nous créerons notre propre image pour regrouper notre configuration de manière plus réutilisable et plus facile à versionner indépendamment Yocto.

J’ai l’habitude de regrouper les modifications que j’apporte à «`local.conf`» vers le début de ce fichier en les encadrant clairement pour pouvoir identifier facilement mes actions. En voici un exemple :

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

MACHINE = "qemuarm"
DL_DIR = "${TOPDIR}/../downloads"
SSTATE_DIR = "${TOPDIR}/../sstate-cache"

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

Bien sûr dès que la complexité des modifications dépasse quelques lignes, il devient préférable de les suivre par l’intermédiaire d’un système de gestion de versions comme `git`.

Les variables renseignées ici sont les suivantes :

- **`MACHINE`** : comme on l’a vu dans la séquence précédente, ceci permet de sélectionner la plateforme cible. Ici, une émulation de carte ARM par Qemu.
- **`DL_DIR`** : lorsque Yocto télécharge des packages de fichiers sources pour les compiler, il les stocke dans ce répertoire en vérifiant au préalable s’ils n’ont

pas déjà été chargés. Par défaut il s'agit du sous-dossier «`dowloads/`» dans le répertoire de compilation (variable `TOPDIR`). J'ai inséré ici un «`../`» pour remonter le répertoire de stockage des fichiers téléchargés un cran au-dessus, à côté de `poky/` et de `build-qemu/`. L'intérêt sera de conserver ce répertoire pour le mettre en commun avec d'autres *builds* pour d'autres cibles.

- **`SSTATE_DIR`** : comme `DL_DIR`, il s'agit d'un répertoire de stockage. Lorsque Yocto progresse dans ses compilations il stocke des fichiers temporaires, des fichiers objets, etc. dans ce dossier. Il peut ensuite les retrouver pour accélérer (très sensiblement) les *builds* suivants. Comme précédemment, j'ai remonté le répertoire d'un «`../`» par rapport à sa situation par défaut pour le partager avec d'autres *builds* éventuels.
- Notons également la présence d'une autre variable qui peut s'avérer utile dans certaines situations : **`BB_NUMBER_THREADS`**. Yocto est très gourmand. Autant en place mémoire ou disque qu'en ressource CPU. Lorsqu'un *build* s'exécute, tous les coeurs de processeur sont sollicités à 100%. Si on doit utiliser la machine de compilation pour un autre travail pendant la compilation, le système va manquer sérieusement de fluidité. On peut donc restreindre le nombre de jobs que `bitbake` lance simultanément pour le rendre moins agressif, en indiquant par exemple «`BB_NUMBER_THREADS="2"`» pour se limiter à deux tâches en parallèle au prix d'un léger allongement du temps de compilation.

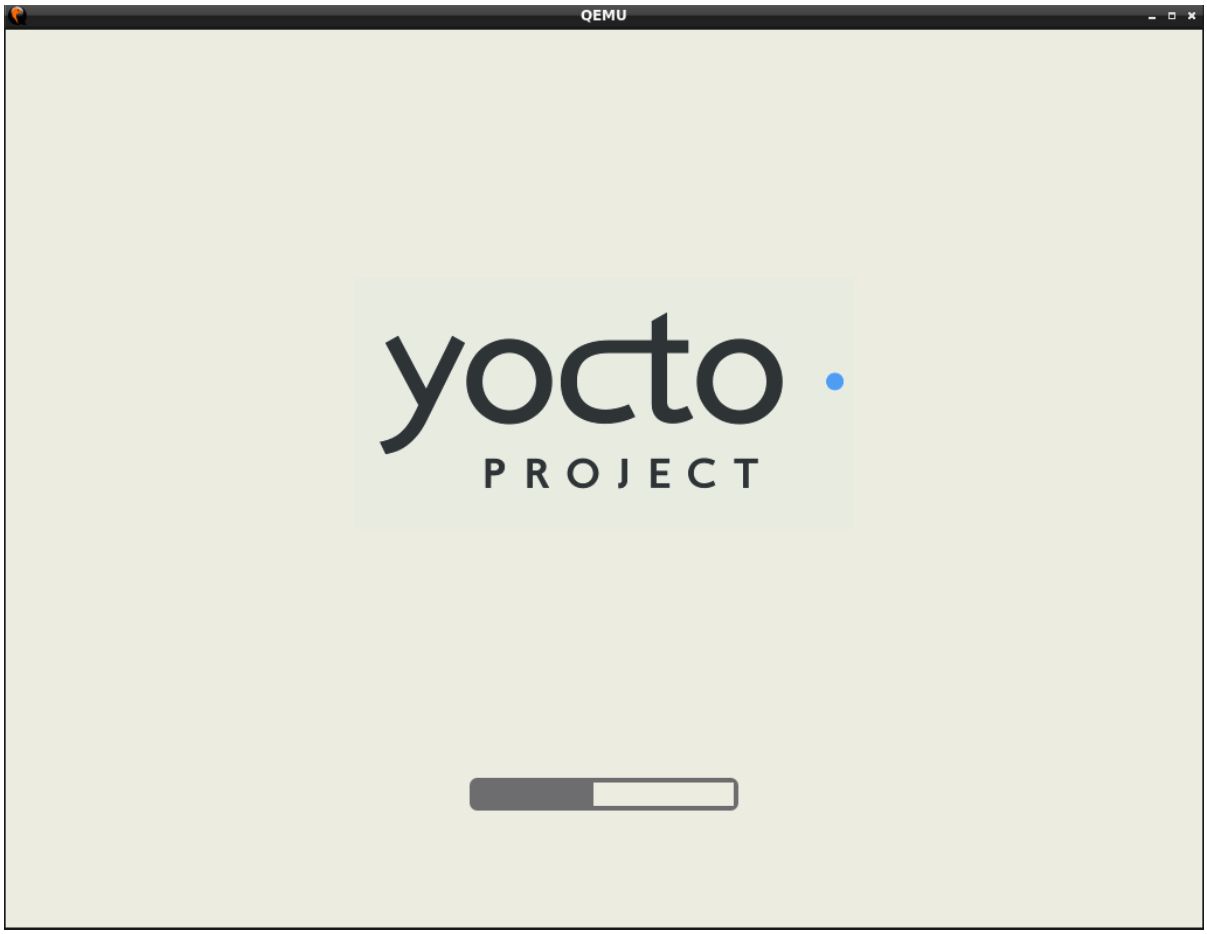
Nous pouvons relancer une génération d'image avec :

```
[build-qemu]$ bitbake  core-image-base
```

La cible «`core-image-base`» est un peu plus riche que «`core-image-minimal`» que nous avons employée dans les séquences précédentes. Une fois la compilation terminée, on démarre l'émulateur avec :

```
[build-qemu]$ runqemu  qemuarm
```

La première différence avec l'image minimale est l'apparition d'un *splashscreen*, une barre de progression graphique pendant le démarrage comme nous le voyons sur la figure II.1-1.



Le nombre de services démarrés au *boot* semble également un peu plus important qu'auparavant :

```
Poky (Yocto Project Reference Distro) 3.0.1 qemuarm /dev/ttyAM

qemuarm login: root
root@qemuarm:~# ps
  PID USER       VSZ STAT COMMAND
    1 root      1392 S    init [5]
    2 root         0 SW   [kthreadd]
    3 root         0 IW<  [rcu_gp]
   [...]
  163 root      2336 S    /sbin/udevd -d
  177 root         0 IW   [kworker/u2:2-ev]
  267 messageb  2336 S    /usr/bin/dbus-daemon --system
  271 rpc       1836 S    /usr/sbin/rpcbind
```

```
  276 rpcuser   2456 S    /usr/sbin/rpc.statd
  296 root         0 IW<  [kworker/u3:1-xp]
  297 root         0 SW   [lockd]
  299 root         0 SW   [nfsd]
  300 root         0 SW   [nfsd]
  301 root         0 SW   [nfsd]
  302 root         0 SW   [nfsd]
  303 root         0 SW   [nfsd]
  304 root         0 SW   [nfsd]
  305 root         0 SW   [nfsd]
  306 root         0 SW   [nfsd]
  308 root      2544 S    /usr/sbin/rpc.mountd
  313 root      2244 S    /sbin/syslogd -n -O /var/log/message
  316 root      2244 S    /sbin/klogd -n
  322 avahi     2852 S    avahi-daemon: running [qemuarm.local
  323 avahi     2852 S    avahi-daemon: chroot helper
  332 root      2404 S    {start_getty} /bin/sh /bin/start_get
  334 root      2244 S    /sbin/getty 38400 tty1
  363 root      2624 S    -sh
  493 root      2332 R    ps
root@qemuarm:~#
```

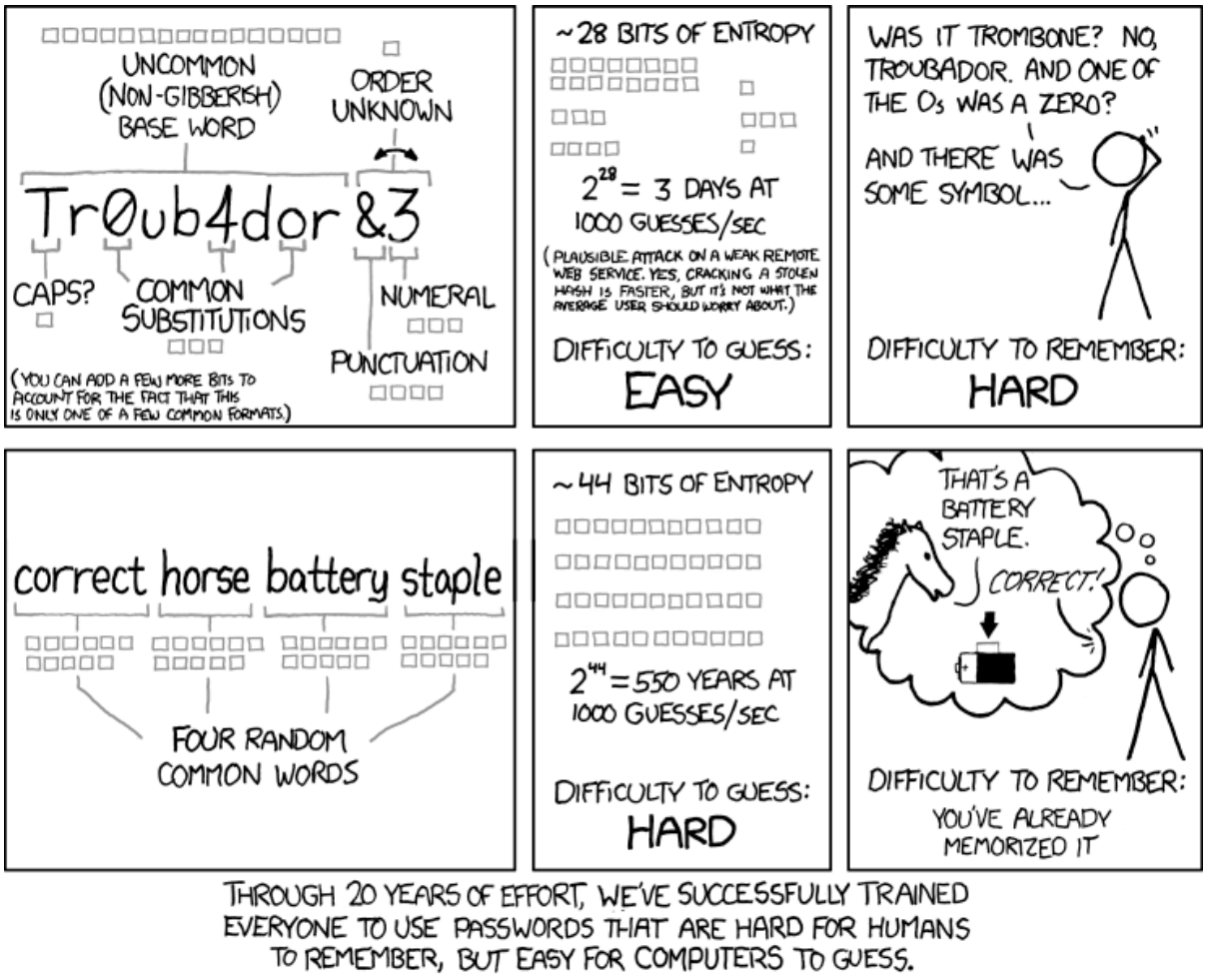
Utilisateurs et mots de passe

Comme auparavant nous pouvons nous connecter sous l'identité «`root`» **sans mot de passe**. Ceci est évidemment très dérangeant pour des raisons de sécurité :

- L'accès au compte *root* est le dernier bastion de la sécurité du système. Dans de nombreux cas on interdit purement et simplement l'accès direct à ce compte en n'acceptant que des connexions d'utilisateurs classiques, parmi lesquels certains – appartenant à un groupe spécifique – seront autorisés à utiliser «`sudo`» pour passer des commandes nécessitant les privilèges de l'administrateur. Si l'on tolère la connexion avec l'identité *root*, il faut au moins qu'elle soit protégée par un mot de passe solide.
- Il n'est pas possible ici de se connecter avec un autre nom de *login*. Pourtant certaines tâches ne nécessitent pas obligatoirement les privilèges *root* et pourraient être réalisées par un utilisateur courant.

Nous souhaitons donc fixer un mot de passe pour l'administrateur *root* (par exemple «`linux`») et créer un compte utilisateur standard, disons «`guest`», avec le mot de passe «`welcome`».

Bien évidemment ces mots de passe sont des exemples typiques de ce qu'il ne faut pas faire. Ils existent dans des dictionnaires et sont suffisamment courts pour être trouvés par force brute. Un bon mot de passe est **un mot de passe long et facile à mémoriser** (par exemple une association de plusieurs mots sans cohérence entre eux). Inutile de mélanger des lettres majuscules, minuscules et des chiffres, ça n'ajoute que très peu d'entropie à la chaîne de caractères. À mon avis, la meilleure explication est celle de Randall Munroe sur son site XKCD :



Pour ajouter un utilisateur standard et modifier le mot de passe de *root* nous allons intervenir dans le fichier «`local.conf`». Tout d'abord nous devons ajouter une ligne un peu mystérieuse :

```
INHERIT += "extrausers"
```

Elle indique simplement que notre configuration va pouvoir utiliser les services d'une classe définie dans Poky permettant la personnalisation des utilisateurs. On notera la syntaxe «+=» pour indiquer que la chaîne de caractères «`extrausers`» est ajoutée à la liste des classes contenues dans «`INHERIT`».

Pour créer le compte utilisateur «`guest`» on ajoutera la ligne suivante :

```
EXTRA_USERS_PARAMS += "useradd  -P welcome  guest;"
```

La commande «`useradd`» appelée n'est pas celle du système («`/usr /sbin/useradd`» sur la plupart des distributions) mais est fournie par Poky dans l'ensemble des outils du système de développement.

On remarquera que non seulement le mot de passe «`welcome`» est mal choisi parce qu'il est trop simple, mais également parce qu'il utilise les lettres «`w`» et «`m`» qui se trouvent à des emplacements différents suivant le type de clavier (*Azerty* / *Qwerty*). Lorsque l'émulateur «`qemuarm`» démarre il est configuré avec un clavier *Qwerty*. Il nous faut donc taper «`zelco,e`» au lieu de «`welcome`» !

Nous pouvons ajouter un mot de passe pour *root* en invoquant la commande «`usermod`» ainsi :

```
EXTRA_USERS_PARAMS += "usermod  -P linux  root;"
```

Si nous relançons le *build*, nous pouvons tester nos connexions au bout de quelques dizaines de secondes :

```
Poky (Yocto Project Reference Distro) 3.0.1 qemuarm /dev/ttyAM
qemuarm login: root
Password: (linux)
root@qemuarm:~# whoami
root
```

```
root@qemuarm:~# exit
logout

Poky (Yocto Project Reference Distro) 3.0.1 qemuarm /dev/ttyAM
qemuarm login: guest
Password: (welcome)
qemuarm:~$ whoami
guest
qemuarm:~$ su -
Password: (linux)
root@qemuarm:~# whoami
root
root@qemuarm:~# exit
logout
qemuarm:~$ exit
logout
```

Cela fonctionne, mais la situation n'est pas très satisfaisante. Bien sûr les mots de passe sont hachés avant d'être inscrits sur la cible. On ne peut en aucun cas être retrouver les mots de passe initiaux, même en étudiant le contenu du fichier «`/etc/shadow`» qui contient les résultats du hachage (à condition évidemment d'utiliser des mots de passe solides, pas les exemples simplissimes que j'ai employés ici et qui se trouvent dans tous les dictionnaires de *crack*).

Néanmoins **les mots en clair se trouvent dans le fichier «`local.conf`»** sur la machine de production. Qu'en est-il de sa sécurité ? Peut-on compromettre la sécurité de tous les produits déjà déployés chez les clients simplement à cause d'un fichier en clair sur le poste d'un développeur ? La réponse est évidemment «non !», il va falloir chiffrer les mots de passe également dans le fichier «`local.conf`» de la machine de production.

Pour cela, les commandes «`useradd`» ou «`usermod`» acceptent une option «`-p`» (à la place de «`-P`») pour fournir un mot de passe haché par la fonction `crypt()` de la bibliothèque C. Pour accéder à cette fonction on peut faire beaucoup de choses différentes, mais j'ai écrit **un petit utilitaire que je vous propose** d'employer.

Le projet s'appelle `password-encryption`, il est sous licence GPL et disponible ainsi :

```
[build-qemu]$ git  clone  https://github.com/cpb-/password-enc
Clonage dans 'password-encryption'...
remote: Enumerating objects: 42, done.
remote: Counting objects: 100% (42/42), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 42 (delta 19), reused 37 (delta 14), pack-reused
Dépaquetage des objets: 100% (42/42), fait.
[build-qemu]$ cd  password-encryption/
[password-encryption]$ make
gcc -Wall -W -c  crypt.c
gcc -o crypt crypt.o -lcrypt
```

On peut hacher un mot de passe simplement ainsi :

```
[password-encryption]$ ./crypt linux
$5$p4pbIZkJ$rdM34E4hsys2v5vSUnj6mbd9jcLQSzKH59rDDDWhvxC
```

L'algorithme utilisé par défaut est *SHA-256*, mais on peut préférer un hachage *MD5* ou *SHA-512*. Une option «`-e`» est bien utile pour nous : elle permet de protéger à l'affichage les caractères «$» en les précédant par des *backslashes* «\». Nous pouvons ainsi copier-coller le hachage pour le placer dans «`local.conf`» sans risque d'interprétation erronée par `bitbake` sous forme de noms de variables.

```
[password-encryption]$ ./crypt  -e  linux
\$5\$IlhPmUHu\$AFPAiidalJ5Llt5YSqKoKXUbiWJjbvtRYT0AvzJp6p7
[password-encryption]$ ./crypt  -e  welcome
\$5\$gFkqOeNU\$BOtEK2RhJIu.MxFT4jqtaYGYdw85fiRbc4hplRWI/63
[password-encryption]$ cd  ..
[build-qemu]$
```

Je modifie donc mon fichier «`local.conf`» pour devenir :

```
# <LOGILIN>
```

```
MACHINE = "qemuarm"
DL_DIR = "${TOPDIR}/../downloads"
SSTATE_DIR = "${TOPDIR}/../sstate-cache"

INHERIT += "extrausers"

EXTRA_USERS_PARAMS += "useradd -p '\$5\$IlhPmUHu\$AFPAiidalJ5l

EXTRA_USERS_PARAMS += "usermod -p '\$5\$gFkqOeNU\$BOtEK2RhJIu.

# </LOGILIN>
```

Attention à ne pas oublier d'encadrer le mot de passe par des *quotes* simples (apostrophes) et l'ensemble de la chaîne par des *quotes* doubles (guillemets). En outre, il faut utiliser l'option «`-p`» de `useradd` et `usermode` en remplacement de «`-P`».

Après recompilation de `core-image-base`, la connexion fonctionne toujours à l'identique mais le fichier «`local.conf`» n'est plus aussi critique en cas d'indiscrétion ou de fuite de données.

Hostname

Une seconde étape de personnalisation, toujours dans le fichier «`local.conf`» concerne le nom de la machine. Visible dans le *prompt* il vaut par défaut «`qemuarm`». Nous pouvons le personnaliser en ajoutant la ligne suivante dans «`local.conf`» :

```
hostname_pn-base-files = "mybox"
```

La syntaxe de cette ligne est plus complexe qu'elle en a l'air. Lorsque `bitake` analyse les recettes et les fichiers de configuration, il lit cette ligne ainsi : «Lors du traitement de la recette dont le nom est `base-files`, configurer la variable `hostname` avec la valeur `mybox`».

En règle générale, dans une affectation de variable, le caractère «souligné» (*underscore*) «_» permet de préciser ou restreindre la portée de l'opération. La chaîne de caractères «_**pn**» (*package name*) peut être imaginée comme une sorte d'opérateur pour restreindre la modification de l'affectation de la variable à la recette dont le nom est «`base-files`».

Comme son nom l'indique, la recette «`base-files`», livrée avec Poky, fournit un certain nombre de fichiers de configuration standards comme `/etc/fstab`, `/etc/host.conf`, `/etc/shells`, etc. En outre elle fournit des méthodes pour personnaliser `hostname`, et proposer un message de bienvenue «*Poky (Yocto Project Reference Distro) 3.0.1 qemuarm /dev/tty1*» avant l'invite de connexion.

Ce message peut être modifié en surchargeant le fichier par défaut, nous le ferons dans une prochaine étape.

Après recompilation, la connexion au système affiche :

```
Poky (Yocto Project Reference Distro) 3.0.1 mybox /dev/ttyAMA0

mybox login: root
Password: (linux)
root@mybox:~#
```

Conclusion

Nous avons édité dans cette séquence le fichier «`local.conf`» se trouvant dans le sous-dossier «`conf/`» du répertoire de compilation. Pour le moment nous avons simplement configuré les utilisateurs (et mots de passe) et le nom de machine, dès la prochaine séquence nous allons ajouter des utilitaires dans notre système embarqué.