

Configuration avancée du BSP

Christophe BLAESS

christophe.blaess@logilin.fr

<https://www.blaess.fr/christophe/>
twitter: @chrisblaess



Ingénierie et formations sur Linux et les logiciels libres
<https://www.logilin.fr>

Extensions de recettes.....	3
Surcharge de fichiers fournis par une recette.....	3
Travaux pratiques : surcharge d'un fichier de recette.....	4
Application : configuration du réseau.....	5
Patch pour un fichier d'une recette.....	7
Travaux pratiques : création d'un patch sur un fichier de recettes.....	8
Utilisation de devtool.....	9
Recherche de la recette fournissant un fichier sur la cible.....	9
Création d'un patch avec devtool.....	10
Travaux pratiques : création d'un patch sur un fichier compilé.....	11
Licences libres.....	12
Bibliothèques libres.....	13
Noyau Linux et device tree.....	14
Choix et configuration du noyau Linux.....	14
Types et versions du <i>kernel</i>	16
Paramétrage du noyau.....	17
Travaux pratiques : configuration du noyau standard.....	18
Travaux pratiques : <i>patch</i> pour le <i>kernel</i>	19
Principe du <i>device tree</i>	20
Travaux pratiques : <i>patch</i> pour le <i>device tree</i>	21

Ce support de formation est distribué sous licence **Creative Commons 4.0**



(Attribution - Partage dans les mêmes conditions).

Vous êtes libres de copier et partager ce document, en mentionnant son origine. Si vous l'intégrez dans un contenu plus vaste, ce dernier devra être distribué avec les mêmes droits.

Ce cours a été rédigé en utilisant des logiciels libres sur système d'exploitation Linux :

- *LibreOffice Writer* pour le support et la mise en page
- *LibreOffice Draw* pour les dessins vectoriels
- *Gimp* pour les images bitmap
- *Excalidraw* (en ligne) pour certains schémas

ILY 8.6

<https://www.blaess.fr/christophe/>

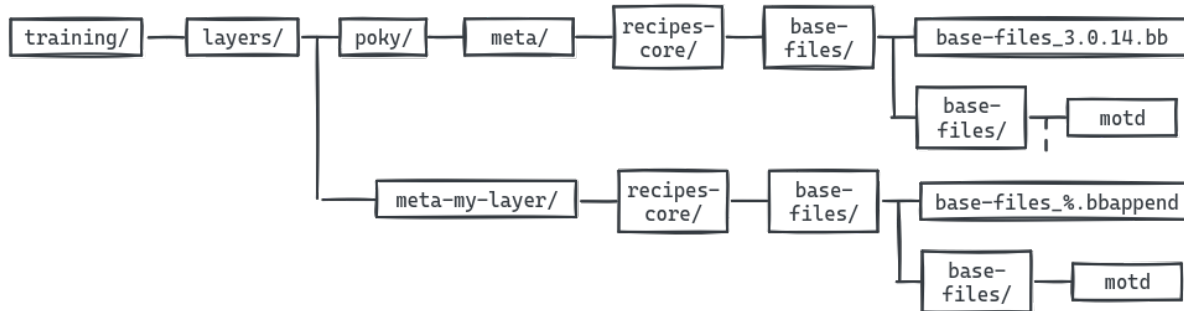
<https://www.logilin.fr>

Extensions de recettes

Surcharge de fichiers fournis par une recette

Pour **surcharger** un fichier de données ou un script fourni par une recette, on écrit un fichier d'**extension de recette** (suffixe `.bbappend`), qui indique à bitbake de chercher les fichiers réclamés par la recette initiale en commençant par notre répertoire.

Par exemple pour remplacer le fichier `motd` (*message of the day*) de la recette `base-files`, on crée l'arborescence suivante :



Le fichier d'extension contient seulement la ligne

```
|| FILESEXTRAPATHS:prepend := "${THISDIR}/${PN}:"
```

Le fichier `motd` réclamé par la recette sera cherché en parcourant notre répertoire avant (`:prepend`) le répertoire original.

Le nom d'un fichier de recette est composé du nom de la recette (par exemple `base-files`) avec des tirets pour symboliser les espaces, puis un caractère souligné (*underscore*) suivi du numéro de version (`3.0.14`).

Si le fichier d'extension est spécifique à une version de la recette on peut l'indiquer dans le nom (exemple `base-files_3.0.14.bbappend`). Le symbole `%` est un caractère générique représentant n'importe quelle chaîne de caractères (comme le `*` du shell).

Travaux pratiques : surcharge d'un fichier de recette

Créez une extension de recette pour personnaliser le fichier `/etc/motd` de la cible.

Le contenu de ce fichier est affiché automatiquement dès qu'un utilisateur se connecte. Il est traditionnellement utilisé pour prévenir les utilisateurs des opérations administratives (maintenance, ajout de services ou de packages, etc.)

Vérifiez que le message s'affiche bien lorsque vous vous connectez sur la cible.

Solution :

```
[build-qemu]$ cd ..
```

```
[build-qemu]$ mkdir -p ../../layers/meta-my-layer/recipes-core/base-files/  
base-files/
```

```
[build-qemu]$ nano ../../layers/meta-my-layer/recipes-core/base-files/bas  
e-files/motd
```

(ajout du contenu désiré)

```
[build-qemu]$ nano ../../layers/meta-my-layer/recipes-core/base-files/bas  
e-files_%.bbappend
```

```
FILESEXTRAPATHS:prepend := "${THISDIR}/${PN}:"
```

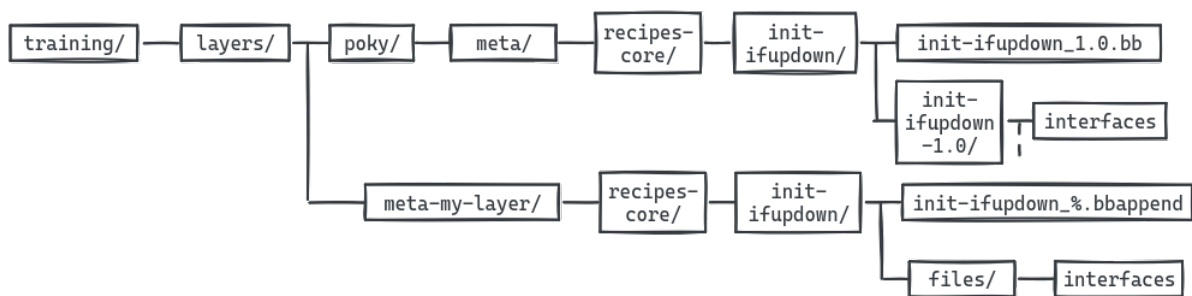
Application : configuration du réseau

Par défaut, Yocto propose une configuration dynamique du réseau s'appuyant sur DHCP.

Nous pouvons néanmoins préférer une configuration statique, notamment si notre cible doit se présenter comme un serveur.

*Créez une arborescence dans le layer pour surcharger un fichier de la recette **init-ifupdown** se trouvant dans **meta/recipes-core** :*

```
[build-qemu]$ mkdir -p ../../layers/meta-my-layer/recipes-core/init-ifupdown/files/
```



Éditez un fichier **interfaces** dans le répertoire nouvellement créé :

interfaces:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 192.168.3.101    ← À ajuster en fonction de votre réseau
    netmask 255.255.255.0
    gateway 192.168.3.254
    dns-nameservers 8.8.8.8
```

Ajoutez un fichier d'extension de recette : ***init-ifupdown_%.bbappend***

Avec la simple ligne :

FILESEXTRAPATHS:prepend:= "\${THISDIR}/files:"

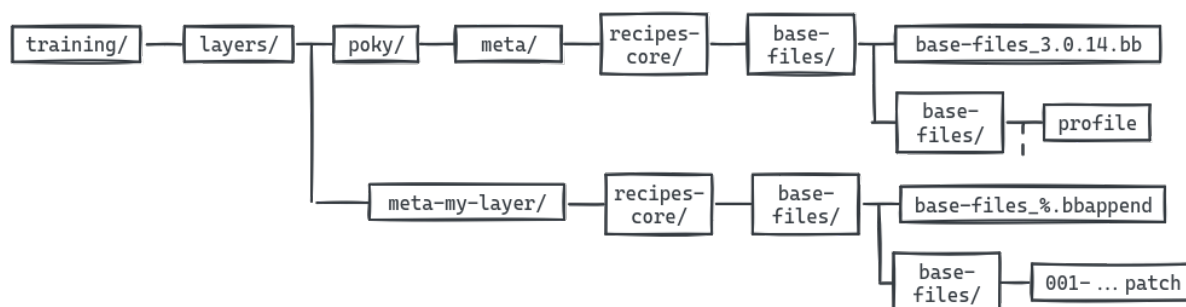
Ajoutez également la ligne suivante dans la recette d'image

IMAGE_INSTALL:append = " resolvconf"

Patch pour un fichier d'une recette

Lorsqu'on souhaite modifier une partie d'un fichier fourni par une recette, on prépare un *patch* qui sera appliqué par bitbake avant la compilation ou l'intégration du fichier.

Le *patch* est fourni dans une extension de recette.



Le nom du *patch* doit être ajouté à la variable SRC_URI dans le fichier .bbappend ainsi :

```
|| SRC_URI += "file://001-...patch"
```

Le patch sera recherché en parcourant les répertoires de FILESEXTRAPATHS.

Travaux pratiques : création d'un patch sur un fichier de recettes

Le fichier `/etc/profile`, fourni par la recette `base-files` configure la variable d'environnement `EDITOR` (l'éditeur préféré par l'utilisateur) avec la valeur `"vi"`.

Toutefois, nous avons installé précédemment `nano`.

Écrivez un *patch* pour la recette `base-files` qui modifie la ligne concernée du fichier `profile`.

Vérifiez que la variable soit bien configurée sur la cible.

Solution :

```
[build-qemu]$ cp -R ../../layers/poky/meta/recipes-core/base-files/base-files .
```

```
[build-qemu]$ cp -R base-files base-files-modified
```

```
[build-qemu]$ nano base-files-modified/profile
```

(modification du contenu de la variable EDITOR)

```
[build-qemu]$ diff -ruN base-files base-files-modified >
../../layers/meta-my-layer/recipes-core/base-files/base-files/001-use-nano-as-default-editor.patch
```

```
[build-qemu]$ nano ../../layers/meta-my-layer/recipes-core/base-files/base-files_%.b bappend
```

```
    SRC_URI += "file://001-use-nano-as-default-editor.patch"
```


Utilisation de devtool

La commande **devtool** permet de nombreuses actions sur les recettes.

Recherche de la recette fournissant un fichier sur la cible

Pour retrouver la recette ou les recettes installant un fichier sur la cible :

```
$ devtool search </path/filename>
```

Pour trouver le fichier .bb d'une recette :

```
$ devtool find-recipe <recipe-name>
```

Pour trouver les fichiers .bbappend concernant une recette :

```
$ bitbake-layers show-append <recipe-name>
```

Exemple :

Recherchez la recette qui installe le fichier /etc/motd (*message of the day*)

Création d'un patch avec devtool

Initialiser les modifications :

```
$ devtool modify <recipe-name>
```

Éditer puis « commiter » les fichiers modifiés

```
$ cd workspace/sources/<recipe-name>
```

```
$ gedit <filename>
```

```
$ git commit <filename> -m "<reason>"
```

```
$ cd ../../..
```

Intégrer les commits sous forme de patches :

```
$ devtool update-recipe <recipe-name> -a <layer>
```

Libérer le répertoire de sources :

```
$ devtool reset <recipe-name>
```

```
$ rm -rf workspace/attic/sources/<recipe-name>
```

Travaux pratiques : création d'un patch sur un fichier compilé

L'éditeur nano affiche au démarrage un message de bienvenue « *Welcome to nano. For basic help, type Ctrl+G* ».

Écrivez une recette pour que le message affiché soit modifié.

Le *patch* doit être produit en se plaçant dans le répertoire contenant l'arborescence des sources de nano et l'arborescence modifiée.

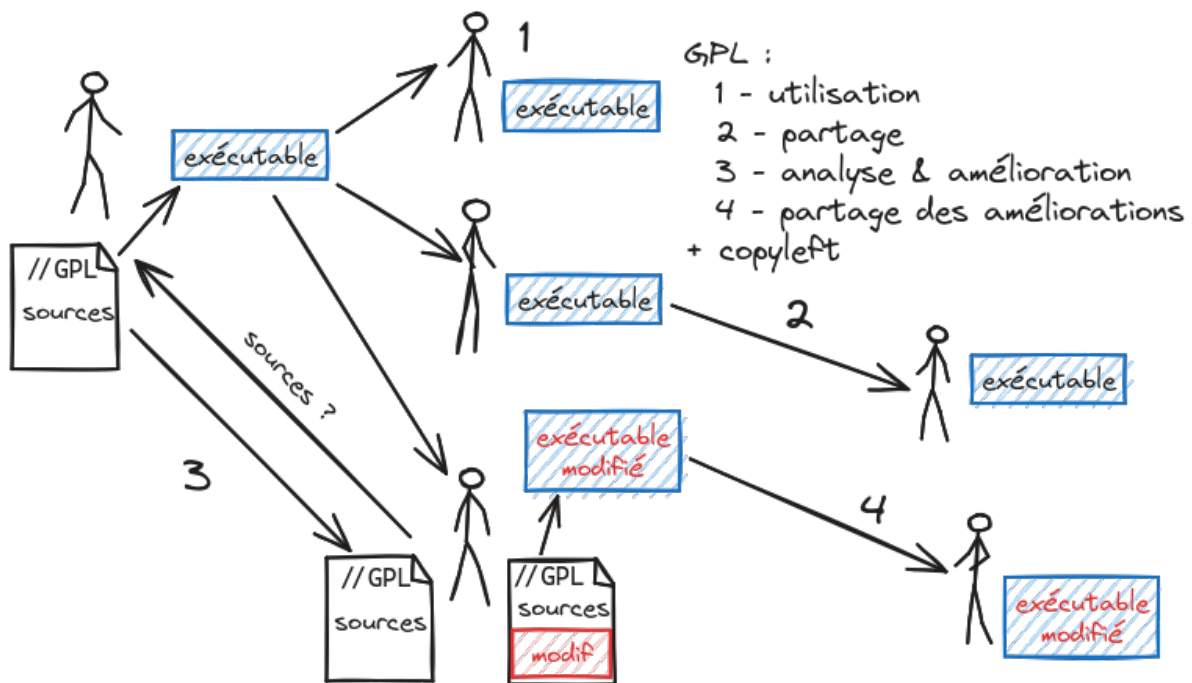
Attention, le message se trouve dans un fichier `.c` et dans une trentaine de fichiers `.po` (pour la traduction). C'est la modification du fichier `.c` qui est intéressante.

NB : l'archive des sources se trouve dans le répertoire `downloads`.

Solution :

```
[build-qemu]$ devtool modify nano
[build-qemu]$ cd workspace/sources/nano/
[nano]$ nano src/nano.c
[nano]$ devtool build-image my-image
[nano]$ git commit src/nano.c -m 'Customize greeting message.'
[nano]$ cd ../../..
[build-qemu]$ devtool update-recipe nano -a ../../layers/meta-my-layer/
[build-qemu]$ devtool reset nano
[build-qemu]$ rm -rf workspace/attic/sources/nano/
```

Licences libres



Un logiciel sous licence GPL v.3 (et v.3 uniquement) ne doit pas être employé dans un environnement qui empêche l'utilisateur de le recompiler et remplacer la version distribuée.

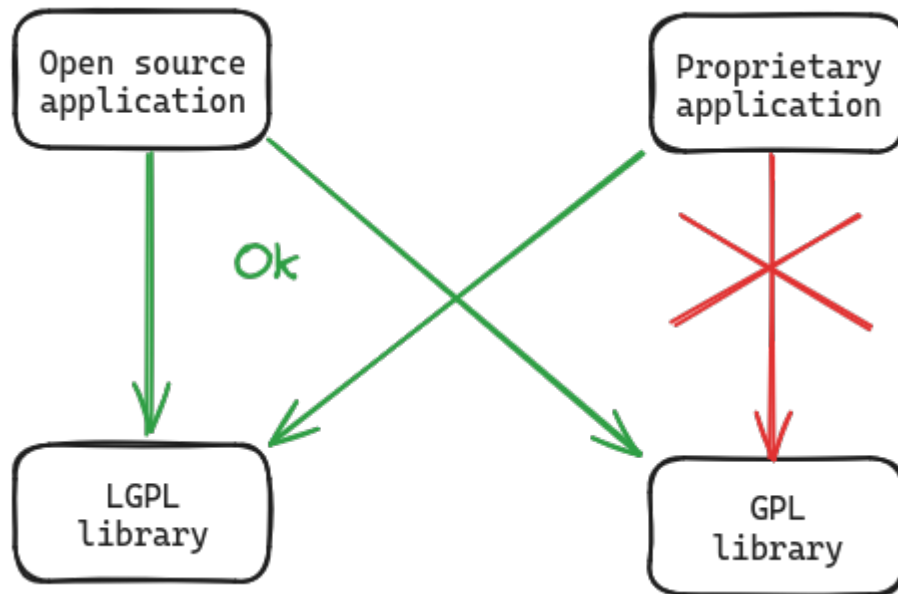
Si on souhaite exclure d'un *build* tous les logiciels sous licences GPLv3, on peut indiquer dans `conf/local.conf` :

```
INCOMPATIBLE_LICENSE = "GPL-3.0* LGPL-3.0*"
```

Bibliothèques libres

La licence LGPL garantit que le code interne d'une bibliothèque (l'implémentation de ses fonctions) est protégée sous les mêmes termes que la GPL.

En revanche, l'utilisation de son API (l'appel de ses fonctions) est possible quelle que soit la licence de l'appelant.



Noyau Linux et *device tree*

Choix et configuration du noyau Linux

Le **noyau** Linux représente le cœur du système d'exploitation, il répartit les ressources matérielles (temps processeur, mémoire, périphériques...) entre les applications.

Influences :

- Unix AT&T – Thomson & Ritchie, 1969
- Unix BSD (*Berkeley Software Distribution*) – University of Berkeley, 1978
- Projet GNU (*Gnu's Not Unix*) – Richard Stallman, FSF (*Free Software Foundation*), 1984
- Linux – Linus Torvalds, 1991.

Le noyau Linux permet de disposer d'un système :

- **multi-tâches** : temps-partagé préemptif classique, temps-réel souple ;
- **multi-utilisateur** : modèle Unix avec utilisateurs et administrateur *root* ;
- **multi-processeur** : support des multiprocesseurs symétriques (SMP) et multicœurs ;
- **multi-architectures** : Intel x86 32/64, ARM, PPC, RiscV, Alpha, PowerPC, etc.

A titre indicatif, le code source du noyau Linux 6.6 contient :

- 1,5 Go de code source (dont 980 Mo pour le sous-système *drivers/*)
- 81.703 fichiers (dont 32.909 dans *drivers/*)
- 32.980 fichiers de code source C (dont 19.660 dans *drivers/*)
- 23.966 fichiers d'en-tête C (*.h*)
- 60 fichiers de code source Rust
- 1.348 fichiers assembleurs
- 2.929 fichiers de compilation Makefile
- 9.462 fichiers de documentation
- 24.777.980 lignes de code source (calculées avec ``sloccount``)
- 18.780 options de compilation

Disponibilité du noyau

La liste de discussion pour les développeurs du noyau est gérée par un robot Majordomo sur `vger.kernel.org`. Les sources du noyau sont sur `www.kernel.org`.

Il existe plusieurs versions simultanées du noyau :

- versions *stable* : par exemple 6.14.3
- versions *longterm* : par exemple 6.6.87
- version *release candidate* : par exemple 6.15-rc3
- version *development* : par exemple next-20250120

Il y a une nouvelle version stable du noyau tous les deux à trois mois environ (nouveaux drivers, protocoles, systèmes de fichiers, améliorations globales, etc.)

Les versions de développement sont accessibles en utilisant le système git.

```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
```

L'interface de programmation **externe** du noyau (API des appels-système) reste toujours stable. Une application n'a pas besoin d'être modifiée ou recompilée entre deux versions du noyau.

L'interface de programmation **interne** du noyau (utilisée par les drivers) peut changer entre deux versions successives du noyau.

Voir aussi

- Les sources du noyau : <http://www.kernel.org/>
- Actualités hebdomadaires sur le développement noyau : <http://lwn.net/kernel/>
- [The Linux Documentation Project](http://www.tldp.org) : <http://www.tldp.org>

Types et versions du *kernel*

Le pseudo-package **virtual/kernel** permet de faire référence au noyau en général, indépendamment de la recette choisie pour l'implémenter.

```
$ ls ../../layers/poky/meta/recipes-kernel/linux
cve-exclusion.inc          linux-dummy          linux-yocto-tiny_6.6.bb
cve-exclusion_6.6.inc      linux-dummy.bb       linux-yocto.inc
generate-cve-exclusions.py linux-yocto-dev.bb    linux-yocto_6.6.bb
kernel-devsrc.bb          linux-yocto-rt_6.6.bb
```

Le *kernel* avec l'extension `-rt` inclut le patch `PREEMPT_RT` qui améliore la prédictibilité de ses temps de réponse.

On peut choisir le type et la version du noyau en déclarant dans `conf/local.conf` :

```
PREFERRED_PROVIDER_virtual/kernel = "linux-yocto-rt"
PREFERRED_VERSION_linux-yocto-rt  = "6.6%"
```


Paramétrage du noyau

Le principe est identique à celui que nous avons vu pour Busybox au chapitre précédent :

Pour invoquer le menu de configuration du noyau :

```
[build-qemu]$ bitbake -c menuconfig virtual/kernel
```

Générer un fichier de différences :

```
[build-qemu]$ bitbake -c diffconfig virtual/kernel
```

```
[...]
```

Config fragment has been dumped into:

```
/home/USER/training/my-build/tmp/work/[...]/fragment.cfg
```

Intégrer le fichier de différences dans le layer personnel :

```
[build-qemu]$ recipetool appendsrcfile -w ../../layers/meta-my-layer/  
virtual/kernel tmp/work/[...]/fragment.cfg
```

Si le menu de configuration ne s'affiche pas, il faut remplir la variable `OE_TERMINAL` de `conf/local.conf` avec la chaîne « screen ».

Travaux pratiques : configuration du noyau standard

Attention : pour les machines basées sur Qemu, tous les drivers nécessaires sont préchargés dans le noyau et les modules ne sont pas installés dans le *rootfs*. Il faut ajouter :

```
IMAGE_INSTALL:append = " kernel-modules"
```

Suppression d'une fonctionnalité

Sur le noyau initial, vérifiez que l'entrée `/proc/config.gz` existe (vous pouvez voir son contenu avec « `zcat /proc/config.gz` »).

Dans le menu de configuration du kernel, désactivez l'option suivante :

```
|| General Setup --->
|| < > Kernel .config support
```

Ajout d'une fonctionnalité

Dans la configuration du *kernel*, ajoutez le *driver* pour un module Ethernet via SPI :

```
|| Device Drivers --->
|| [*] Network device support --->
||   [*] Ethernet driver support --->
||     [*] WIZnet devices
||     <M> WIZnet W5100 Ethernet support
```

Après recompilation, vérifiez sur la nouvelle cible que « `modprobe w5100` » fonctionne.

Solution :

```
[build-qemu]$ nano ../../layers/meta-my-layer/recipes-core/images/my-  
image.bb
```

Ajout de la ligne suivante :

```
|| IMAGE_INSTALL:append = " kernel-modules"
```

```
[build-qemu]$ bitbake -c menuconfig virtual/kernel
```

(Modification des options)

```
[build-qemu]$ bitbake -c diffconfig virtual/kernel
```

```
[build-qemu]$ recipetool appendsrcfile -w ../../layers/meta-my-layer/  
virtual/kernel tmp/work/qemuarm-poky-linux-gnueabi/linux-yocto/[...]  
/fragment.cfg
```

```
[build-qemu]$ bitbake -c cleansstate virtual/kernel
```

```
[build-qemu]$ bitbake my-image
```

Travaux pratiques : *patch* pour le *kernel*

Déterminez la version de noyau s'exécutant sur la cible, et en examinant les recettes mises en œuvre, téléchargez les sources correspondantes (ou extrayez-les du répertoire `downloads/`)

Dupliquez l'arborescence pour pouvoir modifier le noyau.

Éditez le fichier `init/main.c` du noyau et ajoutez, à la fin de la dernière fonction du fichier – `kernel_init_freeable()` – un message avec la fonction suivante :

```
printk(KERN_INFO "My message\n");
```

Ajoutez un *patch* dans le répertoire `recipes-kernel/linux/linux-raspberrypi` créé lors de la configuration du noyau et ajoutez le nom du patch dans la variable `SRC_URI` du fichier d'extension de recette.

Solution :

```
root@qemuarm:~# uname -a
Linux qemuarm 6.6.75-yocto-standard #1 SMP PREEMPT Fri Feb  7 21:37:44 UTC
2025 armv7l GNU/Linux

[build-qemu]$ tar xf downloads/linux-6.6.75.tar.gz
[build-qemu]$ cd linux-6.6.75
[linux-6.6.75]$ git init
[linux-6.6.75]$ git add -A
[linux-6.6.75]$ git commit -a -m 'Initial state'
[linux-6.6.75]$ nano init/main.c
[...]
[linux-6.6.75]$ git commit init/main.c -m 'Custom message during boot'
[linux-6.6.75]$ git format-patch HEAD~1

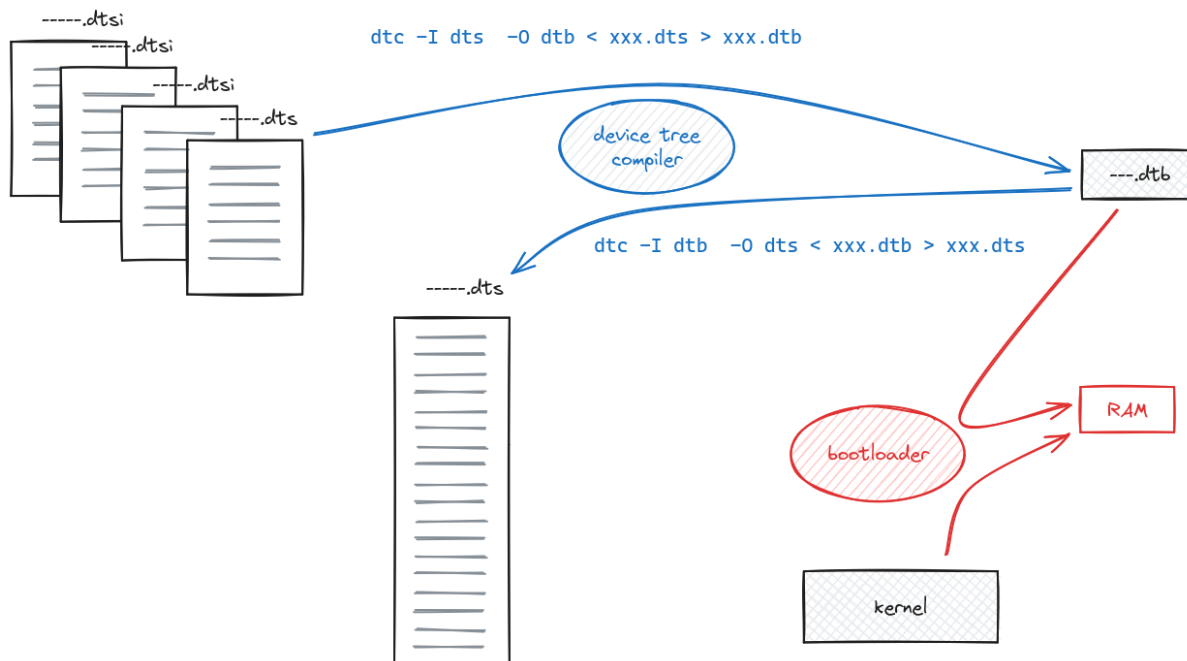
[linux-6.6.75]$ cp 0001-Custom-message-during-boot.patch
../../../../layers/meta-my-layer/recipes-kernel/linux/linux-yocto/

[linux-6.6.75]$ cd ..
[build-qemu]$ nano ../../layers/meta-my-layer/recipes-kernel/linux/linux-
yocto_%.bbappend
|| SRC_URI += "file://0001-Custom-message-during-boot.patch"

[build-qemu]$ bitbake -c cleansstate virtual/kernel
[build-qemu]$ bitbake my-image
```

Principe du *device tree*

Le *device tree* est une description du matériel qui est fournie au noyau sur les architectures ne permettant pas de la déterminer automatiquement (ARM, PowerPC...)



Le fichier au format *Device Tree Source* est compilé par le *Device Tree Compiler* en fichier *Device Tree Blob* (binaire), transmis au noyau par le *bootloader*.

Le fichier `.dts` peut inclure des fichiers `.dtsi` (*Device Tree Source Include*).

Le *Device Tree Compiler* peut « décompiler » un fichier binaire et reconstituer un fichier source `.dts` (en un seul bloc, sans fichier `.dtsi`).

La syntaxe des fichiers `.dts` est assez ardue.

On les trouve dans le répertoire `arch/arm/boot/dts/` du noyau.

Sur un système en fonctionnement, le *device tree* utilisé est exposé (essentiellement sous forme binaire) dans `/sys/firmware/devicetree/`.

Pour en savoir plus :

Pour la syntaxe et la structure du *device tree*, voir <https://www.devicetree.org/>.

Pour les paramètres configurables dans les drivers, voir dans les sources du noyau, le répertoire `Documentation/devicetree/bindings/`.

Travaux pratiques : *patch* pour le *device tree*

Recherchez dans les sources du noyau le fichier source du *device tree* de votre cible.

Modifier le champ "default-trigger" associé à une des leds du système pour lui donner la valeur "heartbeat" ou "timer".

Créez un *patch*, ajoutez-le à votre extension de recette.

Au redémarrage, vérifiez le nouveau comportement de la led.

Solutions :

```
[build-qemu]$ cd linux-6.6.75

[linux-6.6.75]$ nano arch/arm/boot/dts/bcm2711-rpi-4-b.dts
Remplacement de
    linux,default-trigger = "input";
par
    linux,default-trigger = "heartbeat";

[...]
[linux-6.6.75]$ git commit arch/arm/boot/dts/bcm2711-rpi-4-b.dts -m
'Heartbeat led'

[linux-6.6.75]$ git format-patch --start-number 2 HEAD~1

[linux-6.6.75]$ cp 0002-Heartbeat-led.patch ../../layers/meta-my-
layer/recipes-kernel/linux/linux-yocto/

[linux-6.6.75]$ cd ..
[build-qemu]$ nano ../../layers/meta-my-layer/recipes-kernel/linux/linux-
yocto_%.bbappend
|| SRC_URI += "file://0002-Heartbeat-led.patch"

[build-qemu]$ bitbake -c cleansstate virtual/kernel
[build-qemu]$ bitbake my-image
```