



Fabriquer sa rom pdaXrom

Le 6 décembre 2006 à 23:16.

Je voulais depuis un certain temps déjà construire ma propre ROM, basée sur la distribution pdaXrom. Pourquoi faire ? pour y inclure mes paquets préférés, ne pas avoir à tout reconfigurer à chaque installation, mais aussi être plus indépendant vis à vis des auteurs de la distribution et par exemple construire des ROM intermédiaires avant les sorties officielles. La procédure normale pour reconstruire une ROM est de booter sur l'iso de pdaXrom pour i86 et d'utiliser un outil, appelé `builder`. Le problème est que je n'ai aucune envie de booter sur un CD, de dédier une machine à cela, ni même perdre de la ressource et du temps à utiliser vmware. Ce tutorial est donc une manipulation du processus d'origine pour utiliser ces deux mêmes outils mais directement sous Mandriva...

Création du mini-linux

La première phase consiste à extraire de l'image ISO de pdaX86, un l'environnement de compilation complet pour pdaXrom. L'idée est de créer dans un dossier, un mini-linux `chrooté` [1] à partir du contenu de l'image ISO originale. Attention cependant, compiler une ROM prends beaucoup de place, et il faut prévoir au minimum 10Go à l'endroit que vous choisissez de travailler (/home dans mon cas).

```
| mkdir [2] ~/zaurus  
| cd ~/zaurus
```

La première chose à faire est donc de télécharger la dernière version de pdaXrom.iso [3].

```
| wget [4] http://www.pdaxrom.org/download/1.1.0r121/pdaX86/pdaXrom.iso
```

Ceci fait, nous allons monter cette image iso [5]. Elle contient un fichier nommé `boot/rootfs.bin`. C'est un fichier formaté en `squashfs` qu'il va falloir monter à son tour pour obtenir les fichiers de notre mini-linux.

```
| mkdir [2] tmp.iso  
| sudo [6] mount [7] pdaXrom.iso tmp.iso -o loop  
| mkdir [2] tmp.rootfs  
| sudo [6] mount [7] -t squashfs tmp.iso/boot/rootfs.bin tmp.rootfs -o loop
```

Nos fichiers sont maintenant dans `tmp.rootfs`. Nous alors les transférer sur le disque dur physique pour créer le mini-linux. J'utilise `sudo` car certains fichiers contenus dans `rootfs` ne sont manipulables que par `root`.

```
| mkdir [2] pdaXrom  
| sudo [6] rsync [8] -av tmp.rootfs/* pdaXrom/*
```

Les fichiers ainsi copiés, il ne nous reste plus qu'à démonter nos images et à enlever les points de montage :

```
| sudo [6] umount [9] tmp.rootfs  
| sudo [6] umount [9] tmp.iso  
| rm [10] -rf tmp.rootfs tmp.iso
```

Nous allons avoir besoin d'internet dans notre `chroot`. Il faut donc configurer le fichier `pdaXrom/etc/resolv.conf` comme celui du linux hôte (ce fichier est responsable de la résolution des noms de domaine et son contenu pointe sur vos serveurs DNS) :

```
| sudo [6] cp [11] /etc/resolv.conf pdaXrom/etc/resolv.conf
```

Il va maintenant falloir récupérer la dernière version du `builder` de pdaXrom. En effet, si `pdaXrom.iso` contient un pdaXrom pour i86, il ne contient pas pour autant l'outil capable de fabriquer pdaXrom. Cet outil c'est le `builder` et l'on peut simplement le récupérer via la commande `svn` (subversion) :

```
| svn co [12] https://mail.pdaxrom.org/svn/pdaxrom-builder/trunk  
| sudo [6] mv [13] trunk pdaXrom/root/builder
```

Notez le numéro de release subversion logiquement supérieur à 121 (dans mon cas, 165). En effet, depuis la bêta 121 de pdaXrom, les numéros de version sont justement les numéros de release qui s'affiche ici. Cela nous indique donc qu'il a eu 44 modifications depuis la dernière bêta. Si subversion ne vous plaît pas (ou ne marche pas), vous pouvez toujours vous rabattre sur un téléchargement d'une version `tar.bz2` sur le site de pdaXrom.

Il est maintenant temps de lancer notre mini-linux. La commande utilisé est bien évidemment `chroot` auquel nous passons en paramètre le dossier racine (pdaXrom) ainsi que le nom du shell à lancer (`bash`). Les options accolées à `bash` permettent d'ouvrir un shell interactif qui va avant de vous donner la main, exécuter le script `pdaXrom/etc/profile`. Ceci nous permet d'avoir les bonnes variables d'environnement. Le `chroot` est quant à lui lancé via la commande `su -c` pour que ce soit l'utilisateur `root` qui soit utilisé. En effet, pdaXrom a été ainsi développé que tout se passe en tant que `root`. Enfin, la

commande `su` est lancée via la commande `sudo` pour vous éviter de vous logger en tant que `root`. Ouf ! Cela nous donne :

```
| sudo [6] su [14] -c "chroot pdaXrom /bin/bash --rcfile /etc/profile -i"
```

Correction de l'environnement

Normalement vous êtes maintenant dans `pdaXrom`. Il reste cependant quelques bidules à paufiner. Tout d'abord un bug (en tout cas sous `mandriva`) dans le `builder`. Très simple à régler, il s'agit de modifier ligne 46 le fichier `~/builder/configure` pour remplacer `f0` par `f1`. Ceci fait, il faut aussi générer un fichier manquant en lançant la commande suivante :

```
| gdk-pixbuf-query-loaders > /etc/gtk-2.0/gdk-pixbuf.loaders  
# pour plus tard : pango-querymodules > /etc/pango/pango.modules
```

Construction de la tool-chain

Une `tool-chain`, c'est un ensemble d'utilitaires permettant de construire un `linux` à partir de rien. Cela comprends bien évidemment `gcc`, mais aussi `flex`, `bison`, `automake`, etc... Bref, tout ce qu'il faut pour compiler du code binaire pour le `Zaurus` (processeur `ARM`). Le `builder` va faire cela pour nous très simplement. Il suffit tout d'abord de lui demande ce qu'il est capable de fabriquer :

```
| cd ~/builder  
| ./configure | grep [15] xtools
```

Là, on découvre que `pdaXrom` ne se limite pas au `Zaurus`. Pour mieux décrypter cela il faut connaître les races de `Zaurus` (en effet, chaque modèle de `zaurus` est désigné par une race de... chien 🐶) :

Modèle	Race
SL-5000D	Collie
SL-5600	Poodle
SL-C700	Corgi
SL-C750	Shepherd
SL-C760	Husky
SL-C860	Boxer
SL-6000	Tosa
SL-C3000	Spitz
SL-C1000	Akita
SL-C3100	Borzoi
SL-C3200	Terrier

Si vous ne trouvez pas votre modèle dans les configurations, tentez la plus proche. Dans mon cas, c'est `akita`, je vais donc rechercher les configuration pour cette machine :

```
| ./configure | grep [15] akita  
# akita-kernel-2.6-rom  
# akita-kernel-2.6-rom_no_gui  
# akita-kernel-2.6-xtools  
# akita-kernel-2.6-initrd-rom  
# akita-kernel-2.6-initrd-xtools
```

Deux configurations nous intéressent particulièrement : `xtools` (c'est la fameuse `tool-chain`) et `rom` (c'est `pdaXrom` installable). Nous allons donc demander à `configure` de nous créer une `tool-chain` pour `akita` :

```
| ./configure akita-kernel-2.6-xtools ~/myAkita http://mail.pdaXrom.org/src
```

Lorsque `./configure` a terminé, il a créé un dossier `~/myAkita` qui sera notre environnement de travail à partir de maintenant. Nous lui avons aussi spécifié une url pour trouver les sources des paquets à compiler, nous y reviendrons. Maintenant, nous pouvons aller dans notre nouvel environnement pour construire notre `tool-chain` :

```
| cd ~/myAkita  
| make [16] virtual-xchain_install
```

Au fur et à mesure de la progression, le builder va télécharger les sources de chacun des éléments à compiler (gcc, automake, etc...). Ces sources sont stockés dans `~/myAkita/src`. Le problème est que parfois... souvent... le site de pdaXrom tombe en panne. Le téléchargement ne peut donc se faire. Pour régler cela, il suffit d'indiquer au builder de ne plus aller chercher les tarballs sur le site de pdaXrom mais sur l'URL d'origine. Pour se faire, il suffit de supprimer le fichier `~/myAkita/.clone` et de relancer le `make world`.

Si cependant, même le site d'origine ne fonctionne pas, la solution ultime consiste simplement à copier le nom du fichier que lui builder tente de télécharger (par exemple, `gcc-3.4.6.tar.bz2`), de coller ce nom dans google qui doit logiquement vous renvoyer une page où trouver le paquet (par exemple `http://gcc.fyxm.net/releases/gcc-3.4.6/gcc-3.4.6.tar.gz`). Il suffit alors d'arrêter le builder (CTRL-C) et de taper :

```
cd src
wget [4] http://gcc [17].fyxm.net/releases/gcc [17]-3.4.6/gcc [17]-3.4.6.tar [18].gz
cd ..
make [16] virtual-xchain_install
```

Et la compilation peut alors se poursuivre...

Au bout d'une bonne demi-heure (sur un Sempron 2400+ qui ne fait pas que cela), la tool-chain est compilée. Cela veut dire que dans votre dossier `pdaXrom/opt` a été créé un dossier `cross` qui contient toute la tool-chain de compilation pour le Zaurus. Pour éviter d'avoir à refaire cela, nous pouvons demander à créer une archive de cette toolchain :

```
make [16] archive-toolchain
```

Cela ne va rien enlever de ce qui a été fait mais juste faire un fichier `.tar.bz2` à la racine du compte (en `~` donc).

```
ls [19] ~
# armv5tel-cacko-linux-3.4.6-2.16-2.2.5-softfloat-10.20_06.12.06.tar.bz2 builder
myAkita
```

Nous allons maintenant passer à la compilation de la ROM, mais avant, un peu de ménage s'impose. Cela va juste enlever les artefacts de compilations (dans le dossier `build`). Cela n'enlève ni les sources (qui ne seront donc plus à télécharger), ni la toolchain (en `/opt/cross`).

```
make [16] clean
```

Compilation de la ROM

Le builder de pdaXrom est en réalité un projet développé par la société [pengutronix](#) [20]. Il est basé sur des fichiers de configuration (comme vu plus haut) qui fonctionnent un peu comme ceux d'un kernel linux. Pour compiler la toolchain, nous avons chargé la configuration `akita-kernel-2.6-xtools` en spécifiant un dossier de travail `~/myAkita` et une url pour les sources. Nous avons pour cela utilisé le `./configure` du dossier `builder` (celui récupéré de SVN). Nous allons maintenant reproduire cette procédure pour changer de configuration mais cette fois à partir du dossier `~/myAkita`. Comme tout a déjà été configuré, nous n'avons besoin de donner à `./configure` que le nom de la configuration à exécuter. Tout d'abord, pour connaître ce nom pour un modèle akita :

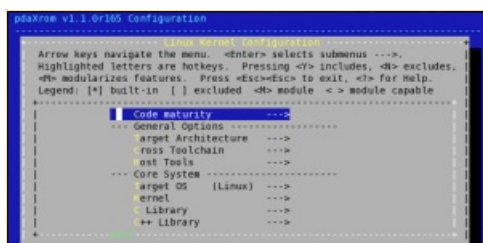
```
cd ~/myAkita
./configure | grep [15] akita | grep [15] rom
# akita-kernel-2.6-rom
# akita-kernel-2.6-rom_no_gui
# akita-kernel-2.6-initrd-rom
```

C'est la première configuration qui nous intéresse

```
./configure akita-kernel-2.6-rom
# Setup for SHARP ZAURUS SL-C1000/C3100 pdaXrom with kernel 2.6, target rom
```

Maintenant, nous pouvons faire une chose très intéressante, à savoir personnaliser pdaXrom. Dans un premier temps, je vous conseil de ne pas le faire et de sauter cette étape. Sinon, il suffit, comme pour un kernel, de taper :

```
make [16] menuconfig
```



Là, s'affiche un menu très conviviale en mode texte. La partie la plus intéressante se trouve dans `Ipk packages` qui nous permet de choisir les packages que l'on cherche à compiler et si oui ou non, ces paquets doivent être installés par défaut. Faites bien attention à ce que vous faites 😊 Une fois terminé, il suffit de sortir en demandant à sauvegarder la configuration. Cette configuration est un fichier texte stocké en `~/myAkita/.config`. Vous pouvez donc en faire une sauvegarde pour la prochaine fois.



Maintenant il est temps de compiler notre ROM et d'aller voir s'il fait beau dehors...

```
| make [16] world
```

Une fois que la compilation est terminée, l'ensemble des nouveaux paquets sont dans le dossier `bootdisk/feed`. Nous pouvons maintenant créer l'image de la ROM :

```
| make [16] image
```

Lorsque c'est fini, il suffit de prendre les fichiers `autoboot.sh`, `kernel.img` et `rootfs.img` qui se trouvent dans le dossier `bootdisk` et de les mettre, comme d'habitude sur une carte SD. De rebooter le Zaurus, maintenir OK pressé pour passer en mode `emergency`, répondre `y` et regarder avec angoisse les blocks défilier. Ensuite le système reboote, et miracle, ça marche !! Je me logue sans souci, lance X sans soucis, et je n'ai plus de bluetooth (le retirer était ma seule modification à la configuration) preuve que je suis bien sur MA rom 😊

Paramétrage du kernel

Paramétrer le kernel permet de comprendre quelques aspects interne du builder. Jusqu'à maintenant nous avons compilé `world`, mais là, nous n'allons compiler QUE le kernel en le modifiant pour nos besoins. Dans la mesure où tout est déjà compilé, le builder ne cherchera pas à reconstruire le reste. La première chose à faire est donc de modifier la configuration du kernel, pour cela, aller dans le dossier `/root/myAkita/build/linux-2.6.16` et taper :

```
| make [16] ARCH=arm menuconfig
```

Là s'affiche le menu classique de configuration d'un kernel. Pour notre test, nous allons ajouter le logo linux au boot du Zaurus. Pour cela, aller dans la section `Device Drivers`, puis `Graphics Support`, `Logo Configuration`, et appuyez sur Espace. Une étoile devrait apparaître ainsi que des sous-options pré-activées. Maintenant faites `exit` plusieurs fois jusqu'à ce que vous soit proposé la sauvegarde, et dites `yes`. Dans la mesure où l'on cherche à compiler un kernel pour ARM et que `menuconfig` vient de compiler des choses pour i86, il est sage à ce stade de taper un `make clean`. Ensuite, il faut retourner dans `~/myAkita` pour relancer une compilation, mais pas de `world` cette fois, du kernel seulement.

```
| cd ~/myAkita
| touch [21] state/kernel.prepare
| make [16] kernel_compile
```

la commande `touch` est utilisé ici pour mettre à la date et à l'heure courante le fichier `state/kernel.prepare`. Cela indique au builder que toutes les étapes suivantes doivent être refaite. Ensuite la commande `make` demande au builder de construire (compile) spécifiquement le kernel. C'est un aspect très intéressant du builder car il permet d'effectuer sur un paquet spécifique une action spécifique sans toucher à ce qui a déjà été fait. La syntaxe générale est `make PAQUET_ACTION`. Pour avoir une liste des paquets utilisable, tapez simplement `make help`.

Une fois la construction terminée, il suffit de reprendre la procédure au `make image` et installer notre nouvelle ROM dans le Zaurus. Et normalement, au démarrage, le logo devrait apparaître.

Personnellement j'utilise ce type de paramétrage pour réduire la taille du noyau à ce qui m'intéresse réellement. PdaXRom a été conçu pour un maximum de gens, moi par exemple je n'utilise pas l'ipv6, je peux donc l'enlever et alléger ainsi le kernel (idem pour bluetooth, irda, etc...). Après chacun peut faire à peu près ce qu'il désire.

Monter la nouvelle ROM

Avant de l'installer sur le Zaurus, il peut être utile de monter la ROM "à la main" pour vérifier qu'elle contient bien ce que nous y avons voulu. Cette même technique peut être appliquée à une version non custom de la ROM. Pour cela nous allons simuler la mémoire flash du Zaurus sur notre PC et y monter notre fichier `rootfs.img` fraîchement créé dans le dossier `bootdisk`. Les manipulations qui suivent peuvent être réalisées au sein du `chroot` mis à part les `modprobe` qui ne fonctionneront qu'à l'extérieur.

Nous allons maintenant créer une "fausse" mémoire flash pour simuler celle du Zaurus et y transférer le fichier `initrd.bin`. Une sorte de flashage en quelque sorte 😊

Pour info, `mtd` veut dire Memory Technology Device. Le module `mtdblock` est là pour offrir une vision périphérique de type "block" à la mémoire flash (qui n'est ni block, ni caractère). On va donc commencer par fabriquer ce périphérique (version majeur 31, si quelqu'un sait pourquoi ?).

```
| cd bootdisk
| modprobe mtdblock
| mknod [22] fausse_flash b 31 0
```

Maintenant nous allons allouer 64m de ram à notre périphérique, la valeur est donc $64 * 1024 = 65535$

```
| modprobe mtdram total_size=65535
```

Ensuite nous allons recopier notre fichier initrd.bin dans la fausse flash

```
| dd [23] if=rootfs.img of=fausse_flash bs=16 skip=1
```

Une fois la copie effectuée, il suffit juste de monter notre nouveau disque en jffs version 2

```
| mkdir [2] rootfs  
| mount [7] -t jffs2 fausse_flash rootfs
```

Et voilà, un petit ls pour vérifier que tout est bien là

```
| ls [19] rootfs
```

Bon, une fois les vérifications terminées, un peu de ménage s'impose

```
| umount [9] rootfs  
rmdir jffs2  
rm [10] -rf rootfs  
rmdir mtdram mtdblock  
rm [10] -rf fausse_flash
```

Conclusion

Pour l'instant notre nouvelle ROM n'est ni plus rapide, ni plus belle, ni plus quoi que ce soit que l'originale. Elle présente cependant deux avantages. Déjà elle nous rends indépendant des releases de pdaXrom du kernel aux applications. De plus, le builder permet d'ajouter d'autres applications et donc d'étendre les paquets générés dans bootdisk/feed. Cela permet de ne plus ré-inventer la roue (comme j'en ai fait jusqu'à maintenant) et donc de bénéficier d'un outil qui semble à première vue aussi puissant que bitbake pour openZaurus. A creuser...

<http://artisan.karma-lab.net/node/1110>
(C) artisan numerique - CC BY-SA

Liens:

- [1] <http://artisan.karma-lab.net/node/1055>
- [2] <http://pwet.fr/man/linux/commandes/mkdir>
- [3] <http://artisan.karma-lab.net/node/1110>
- [4] <http://pwet.fr/man/linux/commandes/wget>
- [5] <http://artisan.karma-lab.net/node/9>
- [6] <http://pwet.fr/man/linux/commandes/sudo>
- [7] <http://pwet.fr/man/linux/commandes/mount>
- [8] <http://pwet.fr/man/linux/commandes/rsync>
- [9] <http://pwet.fr/man/linux/commandes/umount>
- [10] <http://pwet.fr/man/linux/commandes/rm>
- [11] <http://pwet.fr/man/linux/commandes/cp>
- [12] <http://pwet.fr/man/linux/commandes/co>
- [13] <http://pwet.fr/man/linux/commandes/mv>
- [14] <http://pwet.fr/man/linux/commandes/su>
- [15] <http://pwet.fr/man/linux/commandes/grep>
- [16] <http://pwet.fr/man/linux/commandes/make>
- [17] <http://pwet.fr/man/linux/commandes/gcc>
- [18] <http://pwet.fr/man/linux/commandes/tar>
- [19] <http://pwet.fr/man/linux/commandes/ls>
- [20] http://www.pengutronix.de/software/ptxdist/index_en.html
- [21] <http://pwet.fr/man/linux/commandes/touch>
- [22] <http://pwet.fr/man/linux/commandes/mknod>
- [23] <http://pwet.fr/man/linux/commandes/dd>