



- [Accueil](#)
- [A propos](#)
- [Nuage de Tags](#)
- [Contribuer](#)
- [Who's who](#)

Récoltez l'actu UNIX et cultivez vos connaissances de l'Open Source

11 oct 2008

## [Linux embarqué : BusyBox « in a nutshell »](#)

Catégorie : [Embarqué](#) Tags : [lmhs](#)



~~Retrouvez cet article dans :~~ [Linux Magazine Hors série 24](#)

Cet article décrit la mise en place rapide d'un système Linux embarqué autour d'un noyau 2.6 et du composant libre Busybox (<http://www.busybox.net>). Il fait suite aux diverses publications de l'auteur sur ce sujet dont les références sont citées en annexe bibliographique. L'article décrira l'exemple d'un PC x86 mais les concepts sont bien entendu adaptables à d'autres architectures. Le projet buildroot utilisant BusyBox et uClibc sera brièvement présenté en fin d'article sous forme d'un exemple sur architecture ARM9.

La mise en œuvre de Linux pour un système embarqué a été plusieurs fois traitée dans les colonnes de ce magazine ou bien dans la littérature spécialisée (voir bibliographie). Pour réaliser un système Linux embarqué, il convient de rassembler sur un support physique (disque dur ou mémoire flash) les éléments suivants :

- Un noyau Linux adapté à l'architecture matérielle.
- Un ensemble de fichiers exécutables ou scripts nécessaires au démarrage du système.
- Les bibliothèques partagées nécessaires à ces exécutables.
- Divers fichiers et répertoires système tel `/dev`, `/etc` ou `/var`.

La liste finale des composants à installer dépend bien entendu du type d'application prévu pour le système. Cependant, les contraintes matérielles comme la taille de la mémoire flash, de la mémoire vive ou bien le temps de démarrage du système obligent le concepteur à réduire le nombre de commandes disponibles et donc les fonctionnalités. Au final, on peut avoir à disposition un système convenable au niveau applicatif (réalisant la tâche pour laquelle il a été conçu) mais pour lequel les fonctionnalités annexes comme

la configuration ou la maintenance à distance sont réduites à cause du nombre limité de commandes ou de la qualité de ces dernières.

Ce problème a été largement décrit dans l'article « Construction d'un système Linux embarqué », paru dans Linux Magazine ou bien le chapitre 5 de l'ouvrage Linux embarqué, 2ème édition.

## Présentation de la solution BusyBox

Le projet BusyBox (<http://www.busybox.net>) a démarré il y a quelques années dans le sillage du projet Debian. Le but du projet était de fournir un ensemble complet de fonctionnalités « GNU/Linux-like » tout en optimisant l'empreinte mémoire (mémoire vive et flash) en vue de l'installation sur un système cible réduit. En clair, il s'agit de remplacer les commandes classiques (`bash`, `ls`, `cp`, `vi`, etc.) dont la majorité proviennent du projet GNU (<http://www.gnu.org>) par des versions simplifiées mais efficaces. Pour les mêmes raisons de simplification, le principe du démarrage du système (utilisation des run levels ou de scripts tels ceux présents dans ~~etc/rc.d~~) a été largement épuré. Si l'on considère les deux « espaces mémoire » utilisés par un système Linux - soit l'espace noyau et l'espace utilisateur - on peut considérer que BusyBox occupe à lui tout seul l'espace utilisateur, le noyau Linux occupant lui, l'espace... du noyau.

Pour donner une idée de l'occupation sur la flash, un environnement BusyBox sur x86 utilisera moins d'espace que l'interpréteur de commande `bash` (environ 400 Ko contre 600 Ko). A cela, il faudra bien entendu ajouter des bibliothèques partagées, mais leur nombre sera également très réduit.

Au niveau de l'architecture, le principe de BusyBox est simple et se base sur une spécificité du langage C et d'autres langages évolués comme Perl. Dans un programme en C, le premier argument du programme (soit `argv[0]`) contient systématiquement le nom du fichier exécutable correspondant. Si l'on considère le petit programme suivant, soit `argv.c` :

```
#include <stdlib.h>
#include <stdio.h>

main (int argc, char **argv)
{
    printf ("Mon nom est %s\n", argv[0]);
}
```

L'exécution de ce programme donne le résultat suivant :

```
$ ./argv
Mon nom est ./argv
```

Si l'on duplique l'exécutable à l'aide d'un lien symbolique, on obtient :

```
$ ln -s argv argv_bis
$ ./argv_bis
Mon nom est ./argv_bis
```

De ce fait, on peut utiliser un exécutable unique qui pourra remplacer plusieurs fichiers « virtuels » en utilisant simplement le lien symbolique. Au niveau du code, l'exécution de telle ou telle fonction (BusyBox parle d'applet, même si cela n'a rien à voir avec Java) s'effectue simplement en testant le nom du programme. Le principe de BusyBox repose

la-dessus :

- Un exécutable unique nommé Busybox.
- Des liens symboliques portant les noms des commandes remplacées.

Le lecteur curieux, geek ou simplement technophage pourra se reporter aux sources de BusyBox, en particulier au fichier `applets/busybox.c` qui contient la fonction `main` du programme. Bien entendu, il est possible d'étendre BusyBox en ajoutant ses propres applets, comme décrit dans la documentation fournie avec la distribution.

Il faut noter que Le composant BusyBox remplace non seulement les commandes habituelles, mais également la majorité des commandes système couramment utilisées dans un système Linux. On peut citer en exemple :

- La gestion des modules noyau (paquetage `modutils`) y compris pour les versions 2.6.
- La gestion de l'authentification (`login`, `passwd`, etc.).
- La gestion des disques et des partitions (`mount`, `umount`, `fdisk`, `hdparm`, etc.).
- La gestion de l'arrêt/marche (`halt`, `reboot`, etc.).
- La gestion du réseau (`ifconfig`, `route`, `udhcpd`, etc.).

De ce fait, l'installation d'un système sur une base BusyBox est simple et l'on arrive très vite à un résultat fonctionnel, ce qui est un avantage notoire pour la psychologie toujours fragile du développeur :-)

## Compilation de BusyBox

L'archive au format tar+gz est disponible auprès du site <http://www.busybox.net>. Après extraction, on obtient un répertoire correspondant à la version, soit `busybox-1.01` dans notre cas. Même si elle n'est pas parfaite, la documentation présente dans le répertoire `docs` est assez complète.

Le fichier `INSTALL` sur la racine des sources indique la méthode rapide de compilation.

Le fichier `README` donne quelques informations générales sur le composant.

De même, le répertoire `examples` donne quelques exemples de fichiers de configuration dont un fichier `inittab` largement commenté.

La méthode de génération est assez proche de celle du noyau Linux.

1. Configuration des fonctionnalités par `make menuconfig`. L'interface présentée est similaire au `make menuconfig` du noyau Linux. La configuration est sauvée dans un fichier `.config` dont la structure est identique à celle d'un fichier de configuration du noyau.
2. Compilation par `make dep` puis `make`
3. Installation par `make PREFIX=répertoire_installation install`

La figure suivante présente l'écran de configuration suite à un `make menuconfig`.

Dans le menu Build Options, on remarquera la possibilité de construire BusyBox en mode statique (sans utiliser de bibliothèques partagées). De même, le menu permet de sélectionner un compilateur croisé au lieu du compilateur natif utilisé par défaut.

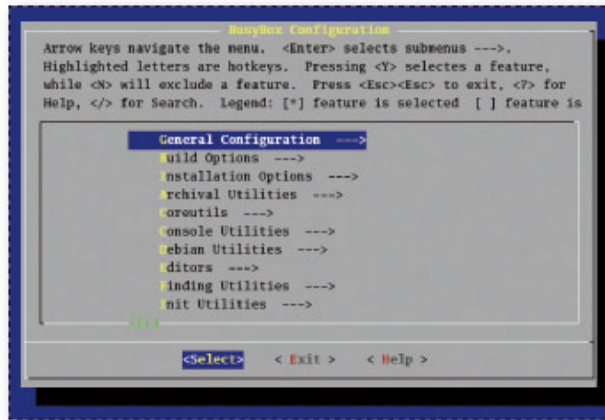


Fig. 1 : Configuration de BusyBox

## Création d'un système BusyBox « in a nutshell »

Un test rapide d'une solution BusyBox minimale est simple à obtenir sur une distribution Linux x86 classique. Dans le cas présent, nous utiliserons une Fedora Core 4 (FC4), basée sur un noyau 2.6. La procédure de création est la suivante :

- 1. Création d'une partition dédiée. Pour cela, on utilisera les commandes `fdisk` et `mkfs2fs`. La partition est ensuite montée sur le répertoire `/mnt/emb` de la FC4.
- 2. Compilation et installation d'un noyau Linux adapté. Pour cela, nous partirons du dernier noyau officiel 2.6 à la rédaction de l'article (soir 2.6.13.4). Cette phase est optionnelle et l'on peut imaginer utiliser dans un premier temps le noyau fourni par la FC4. Cependant, ce noyau est très volumineux (nombreux modules) et n'est pas significatif d'une démonstration de solution réduite typique à un système embarqué. Suite à l'installation d'un nouveau noyau, il sera nécessaire de mettre à jour le fichier `/etc/grub.conf` du programme de démarrage GRUB afin de tester le fonctionnement du noyau puis ajouter la nouvelle entrée correspondant au système BusyBox.
- 3. Création des entrées `/mnt/emb/dev` à l'aide de l'outil `/dev/MAKEDEV` fourni avec la FC4.
- 4. Compilation et installation de BusyBox sur le répertoire `/mnt/emb`
- 5. Installation des bibliothèques partagées nécessaires sur `/mnt/emb/lib`. Pour cela on utilise l'outil `mklibs`.

## Création d'une partition dédiée

Dans notre exemple, nous considérons qu'il est possible de créer une nouvelle partition primaire sur le disque (IDE par défaut).

Le fait d'utiliser un autre type de support (disque SCSI ou clé USB, Compact Flash IDE, DiskOnChip, etc.) change assez peu la démonstration. Dans ce dernier cas, le lecteur pourra se référer à l'article « Linux everywhere » paru dans Linux Magazine (voir

bibliographie).

Dans le cas présent, nous utiliserons la suite de commandes suivante pour créer la partition ~~/dev/hda4~~ occupant 100 Mo sur le disque.

Les commandes à taper par l'utilisateur sont en rouge.

```
# fdisk /dev/hda
```

Le nombre de cylindres pour ce disque est initialisé à 3648.

Il n'y a rien d'incorrect avec cela, mais c'est plus grand que 1024, et cela pourrait causer des problèmes en fonction pour certaines configurations:

- 1) logiciels qui sont exécutés à l'amorçage (i.e., vieilles versions de LILO)
- 2) logiciels d'amorçage et de partitionnement pour d'autres SE (i.e., DOS FDISK, OS/2 FDISK)

Commande (m pour l'aide): p

```
Disque /dev/hda: 30.0 Go, 30011012096 octets
255 têtes, 63 secteurs/piste, 3648 cylindres
Unités = cylindres de 16065 * 512 = 8225280 octets
```

Périphérique	Amorce	Début	Fin	Blocs	Id	Système
/dev/hda1	*	1	765	6144831	7	HPFS/NTFS
/dev/hda2		766	830	522112+	82	Linux swap / Solaris
/dev/hda3		831	1467	5116702+	83	Linux

Commande (m pour l'aide): n

Action de commande

- e étendue
- p partition primaire (1-4)

p

Partition sélectionnée 4

Premier cylindre (1468-3648, par défaut 1468):

Utilisation de la valeur par défaut 1468

Dernier cylindre ou +taille or +tailleM ou +tailleK (1468-3648, par défaut 3648): +100M

Commande (m pour l'aide): w

La table de partitions a été altérée!

## Attention !

Lorsque la partition est créée, il est très souvent nécessaire de redémarrer le système afin que le BIOS la prenne en compte.

Après redémarrage, on peut formater la partition comme suit :

```
# mk2efs -j /dev/hda4
mke2fs 1.37 (21-Mar-2005)
Étiquette de système de fichiers=
Type de système d'exploitation: Linux
Taille de bloc=1024 (log=0)
Taille de fragment=1024 (log=0)
26208 inodes, 104420 blocs
5221 blocs (5.00%) réservé pour le super usager
Premier bloc de données=1
Blocs maximum du système de fichiers=67371008
13 bloc de groupes
8192 blocs par groupe, 8192 fragments par groupe
2016 inodes par groupe
Archive du superbloc stockée sur les blocs:
    8193, 24577, 40961, 57345, 73729
```

```
Écriture des tables d'inodes: complété
Création du journal (4096 blocs): complété
Écriture des superblocs et de l'information de comptabilité du système de fichiers:
complété
```

Le système de fichiers sera automatiquement vérifié tous les 33 montages ou après 180 jours, selon la première év

L'option `-j` indique que l'on crée un système de fichier journalisé, soit au format EXT3. On peut alors monter la partition sur `/mnt/emb`.

```
# mkdir /mnt/emb
# mount /dev/hda4 /mnt/emb
```

Pour assurer le montage automatique à chaque redémarrage, on peut ajouter la ligne suivante au fichier `/etc/fstab` de la FC4.

```
/dev/hda4    /mnt/emb    ext3    defaults    0 0
```

La partition est désormais disponible pour l'installation du nouveau système.

## Compilation et installation d'un noyau Linux adapté

Le noyau fourni avec la FC4 est prévu pour un poste de travail ou un serveur. Il n'est donc pas optimisé pour un système embarqué. L'espace occupé par les modules du noyau (sur `/lib/modules`) est très important, soit plusieurs dizaines de Mo. Le lecteur pressé pourra cependant utiliser le noyau FC4 en première approximation en copiant simplement l'arborescence des modules FC4 sur le répertoire cible.

```
# cd /lib/modules/
# mkdir /mnt/emb/lib/modules
# cp -a -r -f 2.6.11-1.1369_FC4 /mnt/emb/lib/modules
# cp /etc/modprobe.conf /mnt/emb/etc
```

Si l'on désire compiler un noyau adapté, ce qui est préférable, il faut alors suivre la procédure habituelle à partir de l'obtention de l'archive du noyau sur le site <http://www.kernel.org> ou de l'un de ses miroirs.

```
# cd /usr/src/kernels
# tar xjvf /tmp/linux-2.6.13.4.tar.bz2
# cd linux-2.6.13.4
# make xconfig
# make
# make modules_install
# make install
```

La configuration du noyau par `make xconfig` dépend de la configuration matérielle du système. En résumé, une configuration efficace du nouveau noyau se résumera aux points suivants :

- Sélectionner le type de processeur dans le menu Processor type and features.
- Sélectionner uniquement les pilotes et services nécessaires (carte réseau, USB, etc.).
- Sélectionner le support des modules dans le menu Loadable modules support.
- Sélectionner le support IDE, soit ATA/ATAPI/MFM/RLL et Enhanced IDE/MFM/RLL en statique (et non en module) dans le menu Drivers/ATA/ATAPI/MFM/RLL.

- Sélectionner le support de système de fichier EXT3, soit Ext3 journaling file system support en statique (et non en module) dans le menu de configuration File systems.
- Sélectionner le support INITRD soit Initial RAM disk support dans le menu Drivers/Block devices. Ce point est important pour le test du nouveau noyau dans l'environnement complet de la FC4. Par contre, INITRD ne sera pas utilisé pour le système BusyBox, d'où la nécessité des support IDE et EXT3 en statique. De ce fait, l'installation des modules du nouveau noyau ne sera pas nécessaire pour un premier test.

Lorsque le noyau est installé, il est nécessaire d'ajouter l'entrée suivante au fichier `/etc/grub.conf`. Notez que le nom du root filesystem (soit `root=/dev/hda4`) n'est pas modifié puisque l'on teste la FC4 complète.

```
title Fedora Core (2.6.13.4 embedded)
  root (hd0,2)
  kernel /boot/vmlinuz-2.6.13.4 ro root=/dev/hda4
```

Si l'on choisit de tester le noyau Fedora initial, l'entrée sera la suivante :

```
title Fedora Core (2.6.11-1.1369_FC4 embedded)
  root (hd0,2)
  kernel /boot/vmlinuz-2.6.11-1.1369_FC4 ro root=/dev/hda4
  initrd /boot/initrd-2.6.11-1.1369_FC4.img
```

Un redémarrage sur ce nouveau noyau doit conduire à un système fonctionnel. On peut alors ajouter une nouvelle entrée correspondant au système BusyBox. Dans ce cas, le root filesystem utilisé est bien `/dev/hda4` et il n'y a plus de référence à INITRD. Il faut noter que l'on peut également installer le noyau sur la partition cible (soit `/dev/hda4`) ce qui se rapproche plus des conditions réelles d'utilisation. A ce moment-là, il faudra utiliser la commande `root (hd0,3)` au niveau du fichier `/etc/grub.conf`.

## Création des entrées `/mnt/emb/dev`

Cette action est réalisée en utilisant la commande `/dev/MAKEDEV`.

## Compilation et installation de BusyBox

Pour cela, on utilise la procédure décrite au début de l'article et dans le fichier `INSTALL` de la distribution BusyBox. L'installation s'effectue par la commande suivante. On notera que le script d'installation effectue automatiquement la création des répertoires `sbin`, `bin`, `usr` et sous-répertoires sur le répertoire cible.

## Installation des bibliothèques partagées

Si BusyBox n'a pas été compilé en statique (voir le menu Build Options de la configuration BusyBox), il est nécessaire d'installer sur la cible les bibliothèques partagées utilisées. Dans le cas d'une application réelle, nous recommandons l'utilisation des bibliothèques partagées puisque ces dernières seront également utilisées pour les applications ajoutées au système. Dans le cas présent, nous utilisons la `GNU-libc` (ou `glibc`) de la FC4 dont l'image occupe 1,5 Mo sur le disque. Si le système cible nécessite

une empreinte mémoire plus faible, il est également possible d'utiliser la bibliothèque ~~uClibc~~ (<http://www.uclibc.org>) en remplacement de la ~~glibc~~. La taille de ~~uClibc~~ est en moyenne 5 fois plus faible que celle de la ~~glibc~~. Le site du projet ~~uClibc~~ héberge également le projet buildroot (<http://buildroot.uclibc.org>) qui permet de construire très facilement une chaîne de compilation croisée ~~uClibc~~ et une image de système cible basée sur BusyBox et ~~uClibc~~. L'utilisation de buildroot sera décrite à la fin de cet article. Pour faciliter la tâche et optimiser la taille de la ~~glibc~~ sur la cible, nous utilisons le programme mklibs. Ce programme, issu du projet DEBIAN, est disponible sur <http://packages.debian.org/unstable/devel/mklibs>. L'installation de cet utilitaire est très simple :

```
$ tar xzvf mklibs_0.1.17.tar.gz
$ cd mklibs
$ ./configure
$ make
# make install
```

Lorsque l'utilitaire est installé, on peut l'utiliser sur la distribution BusyBox de la manière suivante :

```
# cd /mnt/emb
# mkdir lib
# mklibs -v -d lib bin/*
```

La dernière ligne indique à mklibs d'extraire les bibliothèques partagées utilisées par les exécutables sur répertoire ~~bin~~ (soit en fait ~~bin/busybox~~) et de copier les images optimisées de ces bibliothèques dans le répertoire ~~lib~~. L'optimisation consiste à réduire la taille du fichier .so en copiant uniquement les fonctions utilisées par les exécutables sélectionnés (dans notre cas ~~bin/busybox~~). De ce fait, l'ajout de nouveaux exécutables sur la cible ou la modification de ~~bin/busybox~~ nécessitera un nouvel appel à mklibs.

## Test du système BusyBox

Après redémarrage du système et sélection de l'entrée adéquate dans le menu de GRUB, on doit obtenir l'affichage suivant :

```
Linux version 2.6.13.4 (root@localhost.localdomain) (version gcc 4.0.0 200505195BIOS-provided physical RAM map :
 BIOS-e820: 0000000000000000 - 000000000009fc00 (usable)
 BIOS-e820: 000000000009fc00 - 00000000000a0000 (reserved)
 BIOS-e820: 00000000000f0000 - 0000000000100000 (reserved)
 BIOS-e820: 0000000000100000 - 00000000007ff0000 (usable)
 ...
Freeing unused kernel memory: 144k freed
input: ImPS/2 Generic Wheel Mouse on isa0060/serio1
EXT3 FS on hda4, internal journal
Please press Enter to activate this console.
BusyBox v1.01 (2005.10.17-14:42+0000) Built-in shell (ash)
Enter ,help' for a list of built-in commands.
/ #
```

Le système est cependant dans une configuration minimale dégradée, en l'occurrence :

- Système de fichier monté en lecture seule.
- Clavier anglais.



- Pas de modules installés donc pas d'accès au réseau.
- Pas d'authentification des utilisateurs possible.

Dans la suite de l'article, nous allons décrire les différentes étapes pour aboutir à un système plus peaufiné.

## Amélioration du système

Le système mis en place précédemment est une première version très simplifiée. Nous nous proposons dans ce paragraphe de décrire les modifications nécessaires à la finalisation.

### Montage du système de fichier en lecture/écriture

Traditionnellement, le root filesystem est monté en lecture seule par le noyau (voir l'option `ro` dans le fichier `/etc/grub.conf`). Pour monter le système de fichier en lecture/écriture, il faut utiliser la commande `mount`. L'appel à `mount` sera effectué dans le script `/etc/init.d/rcS` exécuté par BusyBox lors du démarrage. On peut également profiter de l'occasion pour monter le système de fichier virtuel `/proc` nécessaire au bon fonctionnement d'un bon nombre de commandes système (`mount`, `ps`, etc.). Sur le système de développement, on doit créer le point de montage soit :

Ensuite, on peut créer le script `rcS` comme suit :

La première ligne correspondant au remontage de la partition en lecture/écriture. La ligne suivante effectue le montage du système de fichier `/proc` et la dernière ligne monte tous les systèmes de fichier décrits dans `/etc/fstab`. Ce fichier contient les lignes suivantes :

```
/dev/hda4 / ext3 defaults 1 1
none /dev/pts devpts mode=622 0 0
none /proc proc defaults 0 0
```

#### Remarques !

1. Il est nécessaire de monter la partition `/proc` avant l'appel à `mount` car l'option `-a` nécessite la présence de `/proc/mounts` pour fonctionner correctement. Cette entrée contient la liste des systèmes de fichier montés.
2. Le script `rcS` doit être rendu exécutable par la commande `chmod +x rcS`.

### Configuration du clavier français

BusyBox utilise les commandes `dumpkmap` et `loadkmap` pour créer et charger les configurations de clavier. Ces configurations sont différentes du système utilisé par les distributions Linux classiques, soit `loadkeys`. Pour utiliser `dumpkmap/loadkmap`, il faut tout d'abord sélectionner ces programmes dans le menu de la configuration de BusyBox comme décrit dans la figure ci-dessous. Pour créer un fichier de configuration, on utilise `dumpkmap` sur le poste de développement.

```
# cd /mnt/emb
# ./bin/dumpkmap > etc/french.kmap
```

Le fichier créé peut ensuite être utilisé par loadkmap au lancement de BusyBox. Pour ce faire, on peut ajouter la ligne suivante au fichier `rcS`.

```
loadkmap < /etc/french.kmap
```

## Configuration du réseau

Nous avons choisi de placer le support de la carte Ethernet dans la partie dynamique du noyau (les modules). Il faut donc copier l'arborescence des modules sur le répertoire cible par la commande suivante :

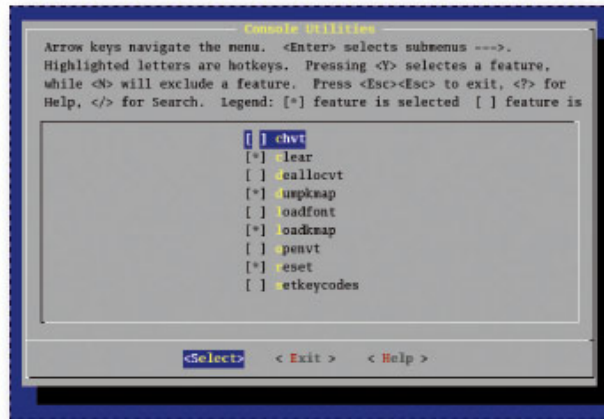


Fig. 2 : Console Utilities

```
# cd /lib/modules/
# mkdir /mnt/emb/lib/modules
# cp -a -r -f 2.6.13.4 /mnt/emb/lib/modules
# cp /etc/modprobe.conf /mnt/emb/etc
```

On peut ensuite initialiser le réseau à l'aide de la commande `ifconfig`.

```
# ifconfig lo 127.0.0.1
# ifconfig eth0 192.168.3.3
```

Si l'on désire utiliser un serveur DHCP, on remplacera la deuxième ligne par un appel à la commande `udhcpd` de BusyBox. Avant cela, il faut copier le script d'exemple fourni avec BusyBox.

```
# cd examples/udhcp
# mkdir -p /mnt/emb/usr/share/udhpc
# cp simple.script /mnt/emb/usr/share/udhpc/default.script
# chmod +x /mnt/emb/usr/share/udhpc/default.script #
```

On peut alors tester le client DHCP par la commande suivante :

```
udhpc
```

Les appels peuvent bien entendu être ajoutés au script `rcS`.

## Authentification des utilisateurs

La version actuelle du système conduit au lancement d'un interpréteur de commande ~~/bin/sh~~ sur la console système. De ce fait, il n'y a pas d'authentification possible des utilisateurs. BusyBox intègre un système classique de login/password similaire à celui d'un système Linux complet. Pour activer cette fonctionnalité, il faut valider les options correspondantes dans le menu Login/Password Management Utilities comme décrit sur la figure 3. Il est également nécessaire de mettre en place un fichier ~~/etc/inittab~~ afin de déclarer le nombre de terminaux (~~tty~~) disponibles. Un exemple commenté de fichier ~~inittab~~ est fourni dans le répertoire ~~examples~~ de la distribution BusyBox. Dans l'exemple fourni, seules les lignes ~~tty4~~ et ~~tty5~~ nécessitent l'authentification (appel à ~~getty~~).

```
#
# Start an «askfirst» shell on the console (whatever that may be)
::askfirst:~/bin/sh
# Start an «askfirst» shell on /dev/tty2-4
tty2::askfirst:~/bin/sh
tty3::askfirst:~/bin/sh
tty4::askfirst:~/bin/sh
# /sbin/getty invocations for selected ttys
tty4::respawn:/sbin/getty 38400 tty5
tty5::respawn:/sbin/getty 38400 tty6
```

Le fichier est à copier sur le répertoire cible :

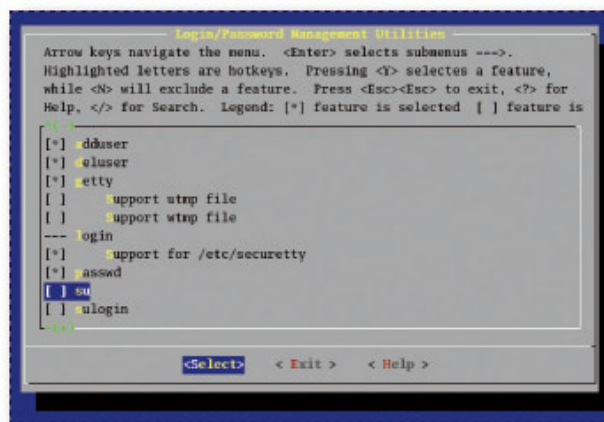


Fig. 3 : Login/Password Management Utilities

Il faut également créer un fichier ~~/mnt/emb/etc/passwd~~ contenant au moins la ligne suivante :

```
root::0:0:Super User:~/bin/sh
```

ainsi qu'un fichier ~~/mnt/emb/etc/group~~ contenant la ligne suivante :

```
root:x:0:
```

Il suffit ensuite d'affecter un mot de passe à l'utilisateur root en tapant la commande ~~passwd~~-root sous BusyBox.

## Utilisation de BUILDROOT

Le projet buildroot est hébergé par le site uclibc.org (<http://buildroot.uclibc.org>). Le but du projet est de fournir un ensemble de procédures permettant de produire facilement un environnement de développement BusyBox et uClibc soit :

- La chaîne de compilation GNU basée sur uClibc.
- La distribution cible basée sur BusyBox, sous forme de fichier image de root filesystem au format EXT2 ou CRAMFS.

De nombreuses architectures sont supportées par buildroot (x86, arm, ppc, etc.). Dans notre cas, nous allons générer une distribution utilisable sur une carte de type ARM9. Il est important de noter que la partie noyau Linux n'est pas prise en compte par buildroot et nous considérons que le matériel dispose d'un noyau Linux fonctionnel et installé. Pour l'installation de buildroot, il est conseillé de partir de l'arborescence Subversion (commande `svn`) disponible auprès du site du projet, soit :

```
$ svn co svn://uclibc.org/trunk/buildroot
```

La ligne précédente provoque la création du répertoire `buildroot`. Il faut alors spécifier les différentes options de compilation. Après l'installation, il suffit de taper `make` pour obtenir l'écran de configuration. Cet écran sera ensuite obtenu grâce à la commande `make menuconfig`. Il faut alors spécifier l'architecture arm dans le menu Target Architecture. Notez que dans le cas de l'architecture x86, il faudra obligatoirement préciser le type de processeur par le menu Target Architecture Variant (i386, i486, i586, i686). Lorsque la configuration est effectuée, il suffit de taper `make` pour générer les différents composants. A l'issue de la compilation, on obtient les composants suivants :

- La chaîne de compilation croisée dans le répertoire `build_arm/staging_dir`.
- Le root filesystem sur `build_arm/root`.
- Une image EXT2 et CRAMFS du root filesystem soit `rootfs.arm.ext2` et `rootfs.arm.cramfs`.

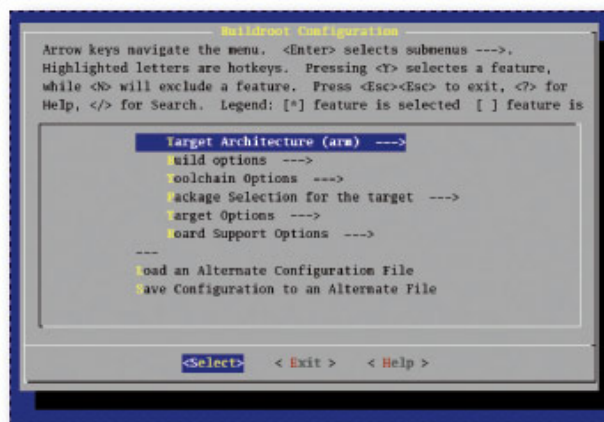


Fig 4. Configuration Buildroot

Dans le cas présent, la carte utilise un système de fichier JFFS2. Il est possible d'indiquer à buildroot de générer une image JFFS2, mais la génération nécessite souvent des options en fonction des caractéristiques matérielles, on peut donc utiliser la fonctionnalité de loopback device du noyau Linux.

```
# mkdir /mnt/rootfs
# mount -t ext2 -o loop rootfs.arm.ext2 /mnt/rootfs
# mkfs.jffs2 -n -l -e 128KiB -d /mnt/rootfs -o rootfs.arm.jffs2
```

L'image ainsi créée peut être copiée sur la mémoire flash de la carte. Il est également possible d'utiliser le root filesystem ~~/mnt/rootfs~~ monté par le protocole NFS. Pour cela, on pourra se reporter au document ~~Documentation/rootfs.txt~~ disponible dans les sources du noyau Linux ou bien au document « NFS-Root mini-HOWTO » disponible sur <http://www.tldp.org/HOWTO/NFS-Root.html>.

## Bibliographie

- L'ouvrage Linux embarqué, 2ème édition, paru aux éditions Eyrolles en septembre 2005 dont la présentation est accessible depuis <http://pficheux.free.fr>
- L'article « Embarquez Linux ! ou Linux everywhere » paru dans Linux Magazine et disponible à l'adresse [http://pficheux.free.fr/articles/lmf/linux\\_everywhere](http://pficheux.free.fr/articles/lmf/linux_everywhere)
- La page de Patrice Kadionik concernant les systèmes embarqués sur <http://www.enseirb.fr/~kadionik/embedded/embedded.html>
- Le site du projet BusyBox sur <http://www.busybox.net>
- Le site du projet uClibc sur <http://www.uclibc.org>
- Le site du projet buildroot sur <http://buildroot.uclibc.org>
- Document « NFS-Root mini-HOWTO » sur <http://www.tldp.org/HOWTO/NFS-Root.html>

~~Retrouvez cet article dans :~~ [Linux Magazine Hors série 24](#)

Posté par Pierre Ficheux ([pficheux](#)) | Signature : Pierre Ficheux | Article paru dans



## Laissez une réponse

Vous devez avoir ouvert une [session](#) pour écrire un commentaire.

« [Précédent](#) [Aller au contenu](#) »

[Identifiez-vous](#)

[Inscription](#)

[S'abonner à UNIX Garden](#)

## • Articles de 1ère page

- [FUSE, développez vos systèmes de fichiers dans l'espace utilisateur](#)
- [SVG - Dessin vectoriel dynamique \(1/2\)](#)
- [Le langage Ada : un peu d'assembleur](#)
- [Placement des contrôles dans une fenêtre en C++ à l'aide de wxWidgets](#)

- [Entrées/Sorties simples sur USB](#)
- [Développement web avancé avec AJAX](#)
- [Quelques techniques d'optimisation en C](#)
- [Supervision avec OCS Inventory NG et GLPI](#)
- [MISC N°41 - Janvier/Février 2009 - Chez votre marchand de journaux](#)
- [Le nouveau modèle objet de PHP5](#)



Actuellement en kiosque :

## • Catégories

- [Administration réseau](#)
- [Administration système](#)
- [Agenda-Interview](#)
- [Audio-vidéo](#)
- [Bureautique](#)
- [Comprendre](#)
- [Distribution](#)
- [Embarqué](#)

- [Environnement de bureau](#)
- [Graphisme](#)
- [Jeux](#)
- [Matériel](#)
- [News](#)
- [Programmation](#)
- [Réfléchir](#)
- [Sécurité](#)
- [Utilitaires](#)
- [Web](#)

## • Articles secondaires

- 30/10/2008

[Google Gears : les services de Google offline](#)

Lancé à l'occasion du Google Developer Day 2007 (le 31 mai dernier), Google Gears est une extension open source pour Firefox et Internet Explorer permettant de continuer à accéder à des services et applications Google, même si l'on est déconnecté....

[Voir l'article...](#)

7/8/2008

[Trois questions à...](#)

Alexis Nikichine, développeur chez IDM, la société qui a conçu l'interface et le moteur de recherche de l'EHM....

[Voir l'article...](#)

11/7/2008

[Protéger une page avec un mot de passe](#)

En général, le problème n'est pas de protéger une page, mais de protéger le répertoire qui la contient. Avec Apache, vous pouvez mettre un fichier `.htaccess` dans le répertoire à protéger....

[Voir l'article...](#)

6/7/2008

[hypermail : Conversion mbox vers HTML](#)

Comment conserver tous vos échanges de mails, ou du moins, tous vos mails reçus depuis des années ? mbox, maildir, texte... les formats ne manquent pas. ...

[Voir l'article...](#)

6/7/2008

[iozone3 : Benchmark de disque](#)

En fonction de l'utilisation de votre système, et dans bien des cas, les performances des disques et des systèmes de fichiers sont très importantes....

[Voir l'article...](#)

1/7/2008

[Augmentez le trafic sur votre blog !](#)

Google Blog Search (<http://blogsearch.google.fr/>) est un moteur de recherche consacré aux blogs, l'un des nombreux services proposés par la célèbre firme californienne....

[Voir l'article...](#)

## • [GNU/Linux Magazine](#)

- - [Lancement des demi-finales Prologin le 24 janvier](#)
  - [GNU/Linux Magazine N°112 - Janvier 2009 - Chez votre marchand de journaux](#)
  - [Édito : GNU/Linux Magazine 112](#)
  - [Les Éditions Diamond adhèrent à l'APRIL !](#)
  - [Nouvelle campagne d'adhésion de l'APRIL !](#)

## • [GNU/Linux Pratique](#)

- - [Linux Pratique N°51 - Janvier/Février 2009 - Chez votre marchand de journaux](#)
  - [Édito : Linux Pratique N°51](#)
  - [Linux Pratique HS N°16 - Janvier/Février 2009 - Chez votre marchand de journaux](#)
  - [Édito : Linux Pratique HS N°16](#)
  - [Linux Pratique HS 16 - Communiqué de presse](#)

## • [MISC Magazine](#)

- - [MISC N°41 : La cybercriminalité ...ou quand le net se met au crime organisé - Janvier/Février 2009 - Chez votre marchand de journaux](#)
  - [Édito : Misc 41](#)
  - [MISC 41 - Communiqué de presse](#)
  - [Les Éditions Diamond adhèrent à l'APRIL !](#)
  - [Misc HS 2 : Cartes à puce, Découvrez leurs fonctionnalités et leurs limites - Novembre/Décembre 2008 - Chez votre marchand de journaux](#)

© 2007 - 2009 [UNIX Garden](#). Tous droits réservés .