Compilation croisée sous Linux et Windows

Pierre Ficheux (pierre.ficheux@openwide.fr)

Mai 2005

Résumé

Cet article décrit la mis en place d'une chaîne de compilation croisée utilisable dans l'environnement Linux x86 ou bien Windows 2000 et XP. Au cours de ce document nous décrirons des tests réels sur une cible Linux ARM mais les concepts décrits restent valable pour une autre architecture type PowerPC ou MIPS.

Introduction

Dans la série d'articles consacrés aux aspects industriels et embarqués de Linux publiés précédemment, nous avons toujours utilisé un environnement Linux x86. Même si cet environnement est très répandu, il est loin d'être le seul utilisé dans ce type d'application. En effet, d'autres processeurs comme l'ARM ou le PowerPC sont parfois mieux adaptés que l'architecture x86.
Cependant, la plupart des développeurs utiliseront un PC x86 (Linux ou Windows) comme poste de travail et il est donc nécessaire de mettre en place un chaîne de développement croisée permettant de développer du code non-x86 sur un PC. Dans cet article, nous allons décrire plusieurs solutions open sources disponibles utilisables sur Linux x86. Ne expliquerons également comment mettre en oeuvre certains de ces outils sur plate-forme Windows en utilisant l'environnement d'émulation CYGWIN. A titre d'exemple, nous mettrons en place et testerons une chaîne de développement pour cible ARM.

La compilation sous Linux

La chaîne de compilation GNU utilise les composants suivants:

- Le compilateur qui constitue le paquetage *gcc*.
- Les outils annexes (assembleur, éditeur de liens, etc.) qui constituent le paquetage *binutils*.
- La *GNU-Libc* qui constitue le paquetage *glibc* Ce paquetage contient en général la bibliothèque de gestion des threads (soit le plus souvent *LinuxThreads*).

Dans le cas d'un système Linux x86, ces paquetages sont installés sur le système sous forme de paquetages RPMS ou DEB dans le cas de la distribution DEBIAN. Dans le cas de la distribution Red Hat, nous aurons par exemple:

```
$ rpm -qv gcc
gcc-3.2.2-5
$ rpm -qv binutils
binutils-2.13.90.0.18-9
$ rpm -qv glibc
glibc-2.3.2-11.9
```

Concernant le paquetage binutils, ce dernier contient des outils utilisés pour la génération et la manipulation des exécutables ou des fichiers objets (.o) intermédiaires. On notera dans ce paquetage le présence de l'assembleur as ou de l'éditeur de liens `ld`.

```
$ rpm -ql binutils
/usr/bin/addr2line
/usr/bin/ar
/usr/bin/as
/usr/bin/gprof
/usr/bin/ld
/usr/bin/nm
/usr/bin/objcopy
/usr/bin/objdump
/usr/bin/ranlib
/usr/bin/readelf
/usr/bin/size
/usr/bin/strings
/usr/bin/strip
...
```

Dans le cas de la compilation croisée, ces différents outils, ainsi que le compilateur, seront exécutés dans un environnement x86 (Linux ou Windows) mais le code généré sera d'un type différent (exemple: ARM). Il est donc nécessaire de compiler le paquetage à partir des source en spécifiant de nouvelles options de génération. Pour ce faire, le script configure inclus dans les paquetages permet de spécifier le type d'architecture sur lequel s'exécute l'outil (option `--host`) ainsi que le type d'architecture cible (option `--target`).
Dans le cas ou les options ne sont pas précisées, le script générera par défaut une configuration pour un exécutable natif, à utiliser dans l'environnement de compilation courant. Voici un exemple pour le paquetage *binutils*.

```
$ ./configure
loading cache ./config.cache
checking host system type... i686-pc-linux-gnu
checking target system type... i686-pc-linux-gnu
checking build system type... i686-pc-linux-gnu
...
```

Par contre si l'on précise les options citées précédemment on obtient:

```
$ ./configure --host=i686-pc-linux-gnu --target=arm-linux
creating cache ./config.cache
checking host system type... i686-pc-linux-gnu
checking target system type... arm-unknown-linux-gnu
checking build system type... i686-pc-linux-gnu
...
```

La génération de la chaîne de compilation croisée est donc réalisable à la main mais elle est souvent fastidieuse car elle peut nécessiter l'application de « patch » sur un ou plusieurs paquetage en fonction des différentes architectures. De ce fait, l'utilisateur non averti risque de disposer d'un chaîne erronée et il est donc conseillé d'utiliser des outils spécialisés.
Dans la suite du chapitre, nous présenterons donc l'outil ELDK (pour *Embedded Linux Development Kit*) développé par *DENX Software* (voir http://www.denx.de/ELDK.html) ainsi que l'outil CROSSTOOL développé par Dan Kegel (voir http://kegel.com/crosstool). Nous précisons

que ces deux outils sont totalement libres et diffusés sous licence GPL. Il n'y a pas non plus de restriction concernant les projets développés avec ces outils.

Mise en oeuvre de l'outil ELDK

ELDK est développé par DENX Sofware, société de consulting Linux située en Allemagne. Le fondateur, Wolfgang Denk, est un professionnel de talent, contribuant à de nombreux projets open source dont RTAI (voir http://www.rtai.org).
Le produit est disponible sous forme de paquetages RPM binaires ou sources. Il est également possible de télé-charger l'image ISO du CDROM contenant les binaires ou les sources. ELDK permet de mettre en place une chaîne de compilation utilisable sur un PC Linux x86 pour des cibles PowerPC, MIPS ou ARM. Cet outil fournit également une distribution que l'on peut utiliser comme *root-filesystem* au travers d'un montage NFS. Cette technique permet de faciliter la mise au point car il n'est pas nécessaire de mettre à jour systématiquement la cible. La version de noyau fournie par ELDK est la 2.4.25 améliorée par Denx principalement pour la plate-forme PowerPC.

L'installation est très simple. Il suffit de télé-charger l'image ISO du CDROM auprès d'un des miroirs du projet soit au choix:

- ftp://mirror.switch.ch/mirror/eldk/eldk/
- http://mirror.switch.ch/ftp/mirror/eldk/eldk/
- ftp://sunsite.utk.edu/pub/linux/eldk/
- http://sunsite.utk.edu/ftp/pub/linux/eldk/
- ftp://ftp.sunet.se/pub/Linux/distributions/eldk/
- http://ftp.sunet.se/pub/Linux/distributions/eldk/
- ftp://ftp.leo.org/pub/eldk/
- http://archiv.leo.org/pub/comp/os/unix/linux/eldk/

La dernière version à ce jour est la 3.1.1, il faut noter que les images ISO n'existent pas forcément pour les anciennes versions comme la 2.1.0. A partir de l'image ISO, il est aisé de graver le CD sur un système Linux, ou même sur une machine Windows. On peut aussi utiliser l'image ISO grâce à la fonction de *loopback device* du noyau Linux. Cette fonctionnalité permet de monter un fichier image comme si il constituait un périphérique physique (exemple: un CDROM au format ISO-9660, une disquette au format VFAT). Elle est disponible en standard sur les noyaux fournis avec les distributions. Sur un noyau compilé, il faudra valider l'option *Block devices/Loopback device support* lors de la configuration du noyau par `make xconfig`.

On peut donc monter le fichier en utilisant la commande suivante:

```
# mount -t iso9660 -o loop mon_image.iso /mnt/cdrom
```

Lorsque l'image (ou le CD) est montée, on peut installer la distribution en exécutant simplement la commande suivante.

```
$ /mnt/cdrom/install -d /opt/eldk_arm
Do you really want to install into /opt/eldk_arm directory[y/n]?: y

Creating directories
Done
```

```
Installing cross RPMs

Preparing...           ########################################### [100%]
   1:rpm               ########################################### [100%]
Preparing...           ########################################### [100%]
   1:rpm-build         ########################################### [100%]
Preparing...           ########################################### [100%]
   1:binutils-arm      ########################################### [100%]
...
```

Dans le cas du PowerPC, on pourra préciser les architectures à installer (ppc_4xx, ppc_6xx, etc.). Par défaut, la procédure installe toutes les architectures.

```
$ /mnt/cdrom/install -d /opt/eldk_ppc ppc_4xx ppc_6xx
```

Remarque:
Il n'est pas nécessaire d'installer ELDK en tant que super-utilisateur (root). Dans le cas ou ELDK est installé en tant que simple utilisateur il est fait bien entendu disposer des droits d'écriture sur le répertoire d'installation. Dans le cas de l'utilisation du *root-filesystem* ELDK par NFS, il sera nécessaire d'utiliser le script `ELDK_MAKEDEV` pour créer les entrées dans le répertoire `/dev` de l'image fournie par ELDK. Si ELDK n'est pas installé en tant que super-utilisateur, il faudra utiliser `ELDK_FIXOWNER` pour configurer correctement l'image NFS comme décrit dans la documentation ELDK sur http://www.denx.de/twiki/bin/view/DULG/ELDKMountingTargetComponentsViaNFS.

Lorsque le paquetage est installé, la chaîne de compilation est utilisable si l'on le chemin d'accès aux outils à sa variable d'environnement PATH comme ci-dessous.

```
$ PATH=/opt/eldk_arm/usr/bin:$PATH
$ export PATH
```

A partir de la on peut utiliser le compilateur et les outils associés.

```
$ arm-linux-gcc -v
Lecture des spécification à partir de /opt/eldk_arm/usr/bin/../lib/gcc-lib/arm-
linux/3.3.3/specs
Configuré avec: ../configure --prefix=/usr --mandir=/usr/share/man
--infodir=/usr/share/info --enable-shared --enable-threads=posix --disable-
checking --with-system-zlib --enable-__cxa_atexit --with-newlib --enable-
languages=c,c++ --disable-libgcj --host=i386-redhat-linux --target=arm-linux
Modèle de thread: posix
version gcc 3.3.3 (DENX ELDK 3.1.1 3.3.3-9)
```

On peut compiler le programme de test classique *Hello World*.

```
$ arm-linux-gcc -o helloword helloworld.c
$ file helloworld
helloworld: ELF 32-bit LSB executable, ARM, version 1 (ARM), for GNU/Linux
2.2.5, dynamically linked (uses shared libs), not stripped
```

La dernière commande indique bien que nous sommes en présence d'un exécutable ARM.

Mise en oeuvre de CROSSTOOL

L'outil CROSSTOOL est quelque peu différent car il est plus complexe à appréhender mais aussi

plus souple. Le complexité vient du fait qu'il n'existe pas de distribution binaire ni de programme d'installation aussi simple que pour ELDK. C'est d'ailleurs tout à fait normal car le but de CROSSTOOL est de fournir à l'utilisateur un ensemble de scripts lui permettant de construire sa chaîne de compilation même dans les cas les plus complexes alors qu' ELDK est limité à l'hôte Linux x86 et aux cibles PowerPC, MIPS et ARM. Avec CROSSTOOL, on choisit donc l'environnement hôte (host), la cible (target), les versions des paquetages à utiliser (gcc, binutils, etc.) mais aussi les différents patches à appliquer aux différents paquetages ou le noyau utilisé et ses patches associés. CROSSTOOL effectue également le télé-chargement des paquetages nécessaire à la génération de la chaîne définie par l'utilisateur.
L'outil CROSTOOL est entièrement écrit en langage script-shell ce qui lui donne un air « old-style » qui n'est pas pour déplaire aux *geeks* chevronnés dont je fais partie!

Pour utiliser CROSSTOOL, il faut tout d'abord installer l'archive télé-chargée depuis le site de Dan Kegel. Dans notre cas nous allons utiliser la version 0.31, dernière à jour au moment de l'écriture de ces lignes. Une fois la distribution installer, une documentation en anglais est disponible dans le fichier `doc/crosstool-howto.html`.
La structure du répertoire `crosstool-0.31` est très simple, les fichiers importants étant localisés directement sur la racine du répertoire.

- Le fichier `all.sh` est le script principal de génération de la chaîne.
- Les fichier de `demo-xxx.sh` comme `demo-i686.sh` sont des exemples fournis dont l'utilisateur peut s'inspirer pour construire sa propre configuration.
- Les fichiers `.dat` permettent de définir des variables d'environnement utilisés par les scripts. Par exemple, le fichier `i686-cygwin.dat` définit des variables indiquant que la cible est CYGWIN.

```
$ cat i686-cygwin.dat
TARGET=i686-pc-cygwin
TARGET_CFLAGS="-O"
```

Les fichiers `.config` correspondent à des configurations du noyau Linux générées par un `make xconfig` ou un `make config` en général et ce pour les différents processeurs supportés (exemples: `i686.config`, `m68k.config`). Ce principe peut être étendu à toute autre système de configuration utilisant un format similaire. Ces fichiers sont utilisés par les fichiers `.dat` cités précédemment.

```
$ cat i686.dat
KERNELCONFIG=`pwd`/i686.config
TARGET=i686-unknown-linux-gnu
TARGET_CFLAGS="-O"
```

D'autres sous-répertoires sont présents et nous pouvons citer.

- Le répertoire `download` qui accueille les paquetages manquants télé-chargés par CROSSTOOL lors de la génération de la chaîne.
- Le répertoire `patches` qui contient lui-même des sous-répertoire correspondant aux différents paquetages utilisables. Chaque sous-répertoire contient les patches à appliquer en fonction de la configuration définie par l'utilisateur. Nous pouvons donner l'exemple du paquetage correspondant à *gcc-3.3.4*.

```
$ ls -w 1 patches/gcc-3.3.4/
gcc-3.3.4-arm-bigendian.patch
```

```
gcc-3.3.4-libstdcxx-sh.patch
gcc-3.3.4-ppc-asm-spec.patch
```

La documentation propose en tant qu'introduction de générer un chaîne de compilation native i686 ce qui n'a pas beaucoup d'intérêt autre que pédagogique. Pour cela il suffit d'utiliser la commande suivante:

```
$ ./demo-i686.sh
```

Le script est extrêmement simple, nous avons au début du fichier les lignes suivantes:

```
#!/bin/sh
set -ex
TARBALLS_DIR=$HOME/downloads
RESULT_TOP=/opt/crosstool
export TARBALLS_DIR RESULT_TOP
GCC_LANGUAGES="c,c++"
export GCC_LANGUAGES

# Really, you should do the mkdir before running this,
# and chown /opt/crosstool to yourself so you don't need to run as root.
mkdir -p $RESULT_TOP
```

Suite à cela, la génération de la chaîne est lancée par la ligne suivante, qui indique que la chaîne en question sera basée sur la configuration i686, le compilateur *gcc-3.4.3* et la *glibc-2.3.4*.

```
eval `cat i686.dat gcc-3.4.3-glibc-2.3.4.dat` sh all.sh –notest
```

Ce petit exemple n'est cependant pas très significatif et nous allons construire un script `demo-at91.sh` permettant de créer une chaîne de compilation croisée Linux x86 vers une architecture ARM ATMEL AT91RM9200. Cette chaîne est similaire à celle fournie dans ELDK.

Au début du fichier, nous indiquons le répertoire dans lequel CROSSTOOL stockera les paquetages télé-chargés:

```
TARBALLS_DIR=`pwd`/downloads
export TARBALLS_DIR
mkdir -p $TARBALLS_DIR
```

Ensuite nous pouvons définir le répertoire destination de la cible.

```
RESULT_TOP=/opt/crosstool
export RESULT_TOP
mkdir -p $RESULT_TOP
```

Comme pour l'exemple précédent, la chaîne pourra traiter les langages C et C++.

```
GCC_LANGUAGES="c,c++"
export GCC_LANGUAGES
```

La chaîne utilisera le noyau 2.4.27, la configuration du noyau étant définie dans le fichier `config-kernel`.

```
KERNELCONFIG=`pwd`/patches/linux-2.4.27/config-kernel
export KERNELCONFIG
```

Le répertoire `linux-2.4.27` n'existe pas dans la distribution CROSSTOOL mais il suffit de l'ajouter. Le répertoire contient les différents patches à appliquer au noyau ainsi que le fichier de configuration du noyau.

```
$ cd patches/linux-2.4.27
$ ls -w 1
01_2.4.27-vrs1.patch
02_2.4.27-vrs1-at91-06102004.patch
03_2.4.27-mkdep+cross-depmod.patch
04_2.4.27-kbd+sram+flash.patch
config-kernel
```

Remarque:
Pour que CROSSTOOL considère le fichier comme un patch, le nom du fichier doit obligatoirement contenir la chaîne de caractère `patch` ou bien le suffixe `.diff`. De même, il est recommandé de numéroter les fichiers de patch en commençant leurs noms par l'ordre d'application (01, 02, etc.). Le lecteur désirant plus d'information pourra consulter le code source du script `getandpatch.sh`.

Si nous revenons à notre script principal, la cible de la chaîne est l'architecture *arm-linux*.

```
TARGET=arm-linux
export TARGET
TARGET_CFLAGS="-O"
export TARGET_CFLAGS
```

La chaîne de compilation correspondante sera installée dans `/opt/crosstool/arm`.

```
PREFIX=${RESULT_TOP}/arm
export PREFIX
```

Enfin nous définissons la liste des paquetages à utiliser.

```
BINUTILS_DIR=binutils-2.15
GCC_DIR=gcc-3.3.4
GLIBC_DIR=glibc-2.3.2
GLIBCTHREADS_FILENAME=glibc-linuxthreads-2.3.2
LINUX_DIR=linux-2.4.27
export BINUTILS_DIR GCC_DIR GLIBC_DIR GLIBCTHREADS_FILENAME LINUX_DIR
```

La génération de la chaîne elle-même est effectuée par la dernière ligne.

```
eval sh all.sh --notest
```

Après avoir lancé la génération par la commande `demo-at91.sh`, nous devons attendre de longues heures avant de récolter les fruits de nos efforts. Afin de suivre le déroulement de l'exécution, il est souhaitable de conserver les traces dans un fichier par la commande:

```
$ ./demo-at91.sh 1>build.log 2>&1 &
```

On peut de temps en temps suivre l'évolution de la compilation par la commande:

```
$ tail -f build.log
```

Lorsque la chaîne est générée avec succès nous obtenons le message suivant dans le fichier.

```
Cross-toolchain build complete. Result in /opt/crosstool/arm.
testhello: C compiler can in fact build a trivial program.
Done.
```

A partir de la, on peut tester la chaîne de compilation comme nous l'avons fait pour ELDK.

```
$ PATH=/opt/crosstool/arm/bin:$PATH
$ export PATH
$ arm-linux-gcc -v
Reading specs from /opt/crosstool/arm/lib/gcc-lib/arm-linux/3.3.4/specs
Configured with: /home/pierre/test/crosstool-0.31/build/arm-linux/gcc-3.3.4-
glibc-2.3.2/gcc-3.3.4/configure --target=arm-linux --host=i686-host_pc-linux-gnu
–prefix=/opt/crosstool/arm --with-headers=/opt/crosstool/arm/arm-linux/include
--with-local-prefix=/opt/crosstool/arm/arm-linux --disable-nls --enable-
threads=posix --enable-symvers=gnu --enable-__cxa_atexit --enable-languages=c,c+
+ --enable-shared --enable-c99 --enable-long-long
Thread model: posix
gcc version 3.3.4
```

On peut dès lors compiler le programme de test *Hello World*.

```
$ arm-linux-gcc -o helloword helloworld.c
$ file helloworld
helloworld: ELF 32-bit LSB executable, ARM, version 1 (ARM), for GNU/Linux
2.4.3, dynamically linked (uses shared libs), not stripped
```

Utilisation de l'environnement CYGWIN

La quasi-totalité des outils présentés dans ce magazine sont utilisable dans un environnement Linux. Cependant, il est clair que c'est loin d'être aujourd'hui l'environnement de développement le plus utilisé. La plupart des outils commerciaux sont disponibles souvent exclusivement sous Windows et la mise en place d'une chaîne de développement sous Linux peut poser des problèmes dans les grandes entreprises pour lesquelles la gestion du parc informatique est centralisée.
La société CYGNUS (liée à Red Hat Software) propose depuis longtemps un environnement sous Windows permettant de porter très rapidement les applications de Linux vers Windows. Le principe est de diriger les appels systèmes normalement destinés au noyau Linux vers une *DLL* (pour *Dynamic Loading Library*) Windows fournie par CYGWIN sous le nom de `CYGWIN.DLL`.

La majorité des outils disponibles sous Linux sont de ce fait disponibles sous CYGWIN, citons entre-autres:

- les commandes Linux à commencer par l'interpréteur de commande `bash`
- la chaîne de compilation GNU
- l'environnement graphique XFree86
- le bureau graphique KDE

De ce fait, il est possible à l'aide de CROSSTOOL de construire une chaîne de compilation croisée

utilisable sous Windows et CYGWIN et permettant de générer du code ARM.

Remarque:
A matériel équivalent, il faut cependant noter que la configuration CYGWIN sera moins performante que la même chaîne utilisée sur un systèmes Linux. De même, il existe encore quelques problèmes dans les adaptations CYGWIN des outils graphique comme KDE. La solution CYGWIN est donc à utiliser en dernier recours.

Installation de l'environnement CYGWIN

La distribution CYGWIN est utilisable sur des environnement Windows récents comme Windows 2000 ou Windows XP. Elle est disponible sur Internet sur le site http://www.cygwin.com. A partir de la page de garde du site, on peut télé-charger le fichier `setup.exe` qui est une application Windows permettant d'installer la suite de la distribution.

Remarque:
Il est est préférable d'installer CYGWIN sur une partition de type NTFS et non VFAT afin d'éviter certains problèmes de droits d'accès aux fichiers. On peut connaître le type de système fichier en affichant les propriétés Windows de la partition en question (utiliser le bouton droit de la souris sur le volume correspondant).

Lorsque l'on double-clique sur l'icône associée au fichier `setup.exe`, on obtient la fenêtre suivante.

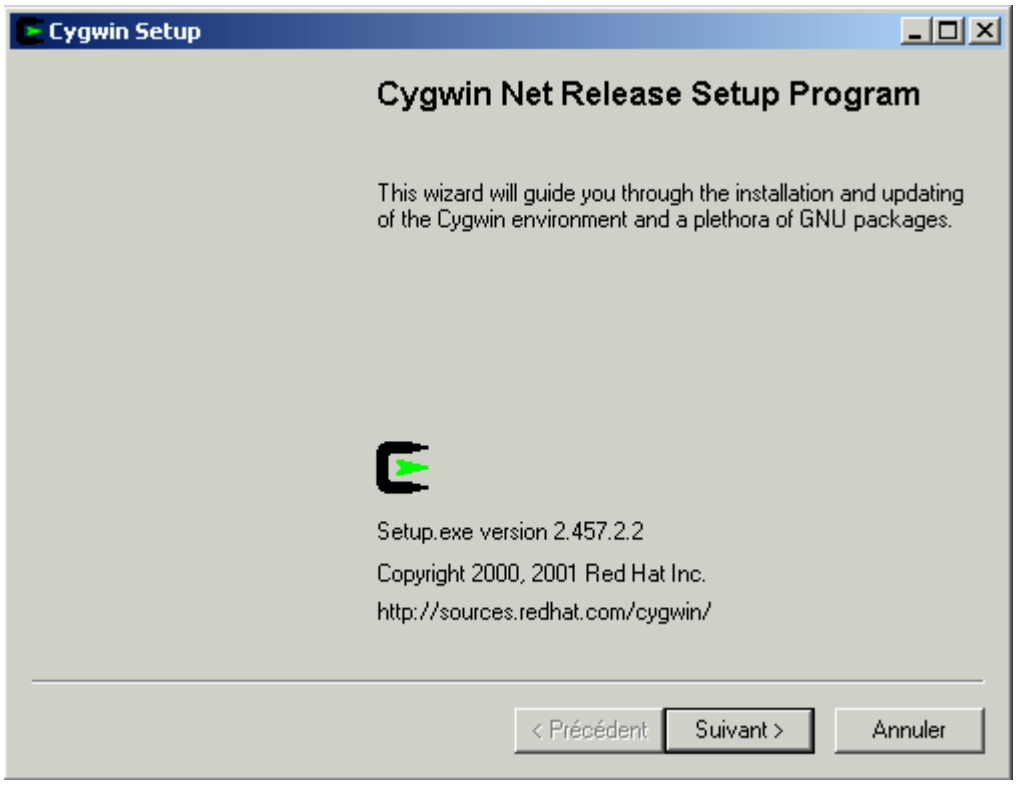

Figure 1: Exécution de `setup.exe`

Lors de la première installation, il est conseillé d'utiliser l'option *Install from Internet*. Les fichiers utilisés sont sauvés sur le disque local ce qui permettra d'effectuer ultérieurement une installation à

partir d'un répertoire local (*Install from Local Directory*) si cela était nécessaire.

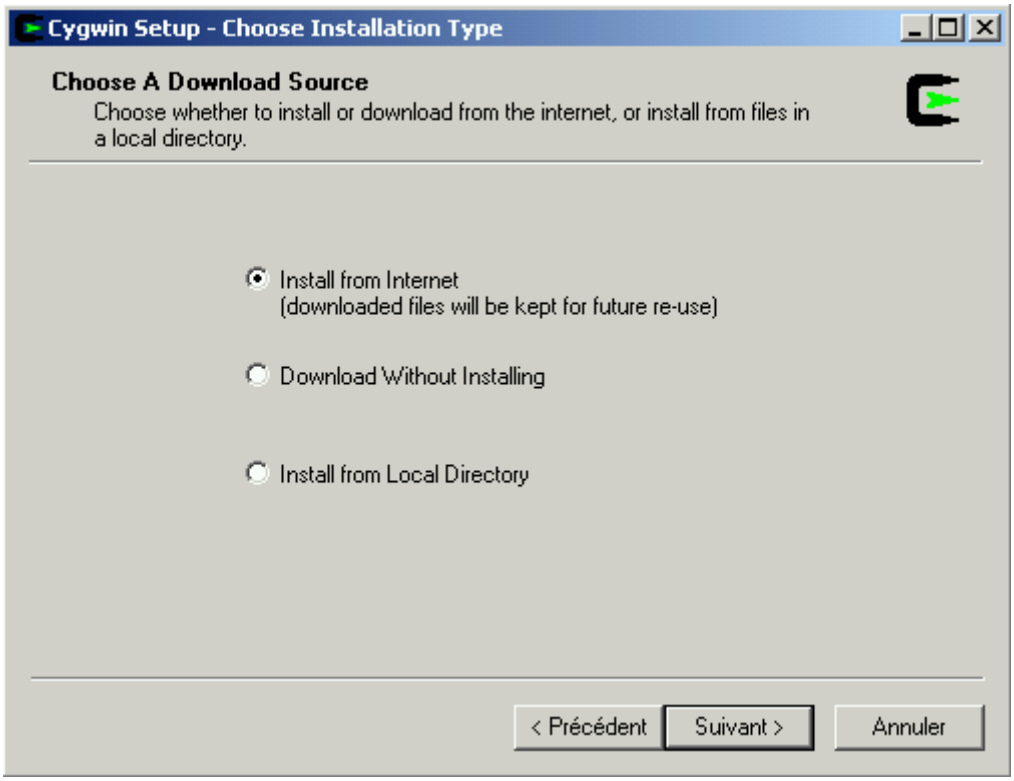

Figure 2: Choix du type d'installation

L'écran suivant sélectionner le répertoire d'installation (soit `c:\cygwin` par défaut).

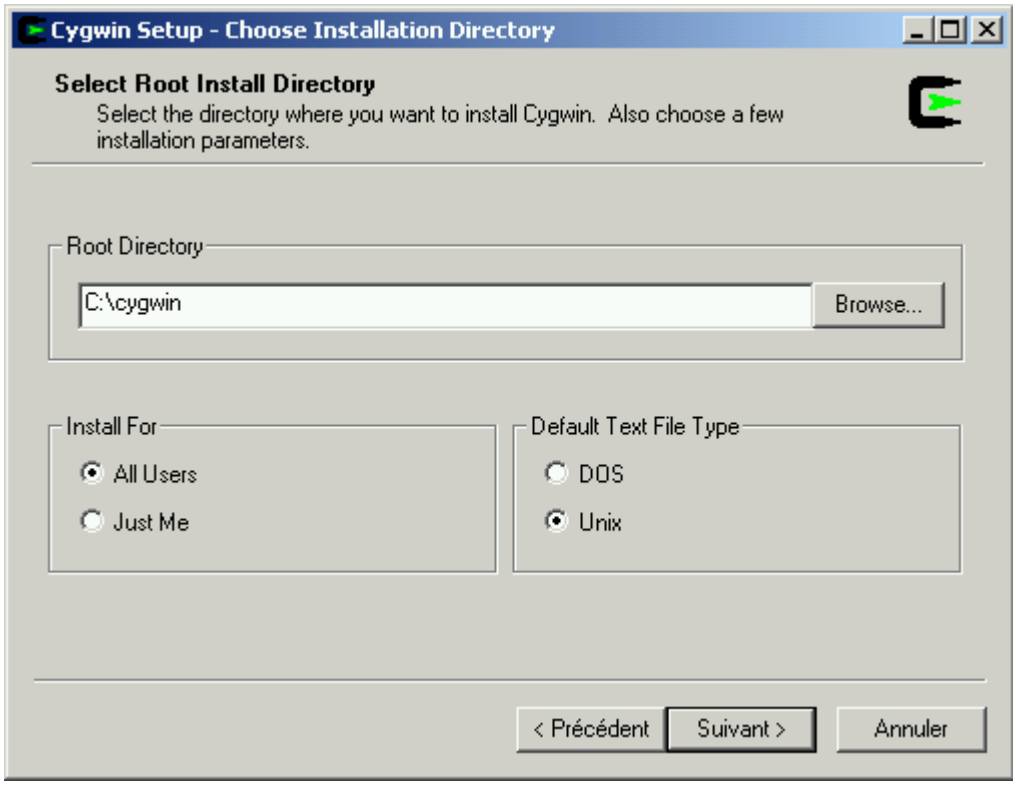

Figure 3: Choix du répertoire d'installation

On sélectionne ensuite un serveur miroir de la distribution CYGWIN permettant le chargement des fichiers.

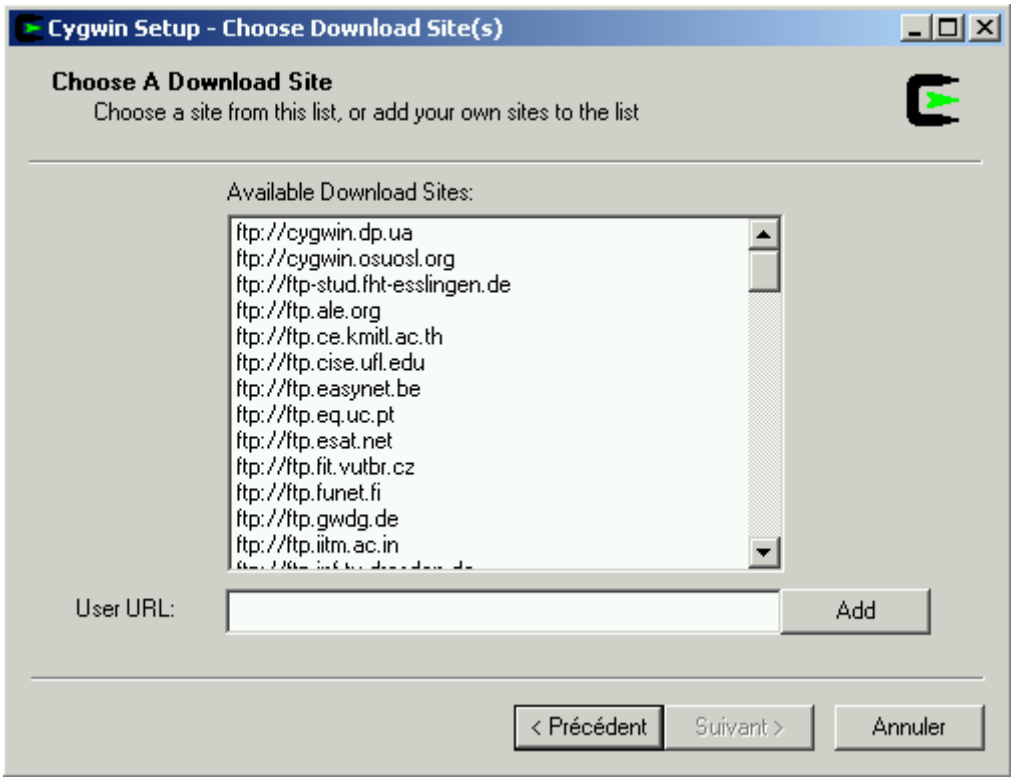

Figure 4: Choix du serveur miroir

Le programme d'installation récupère alors la liste des paquetages et la présente à l'utilisateur. Dans le cas présent nous conseillons d'installer la totalité de la distribution.

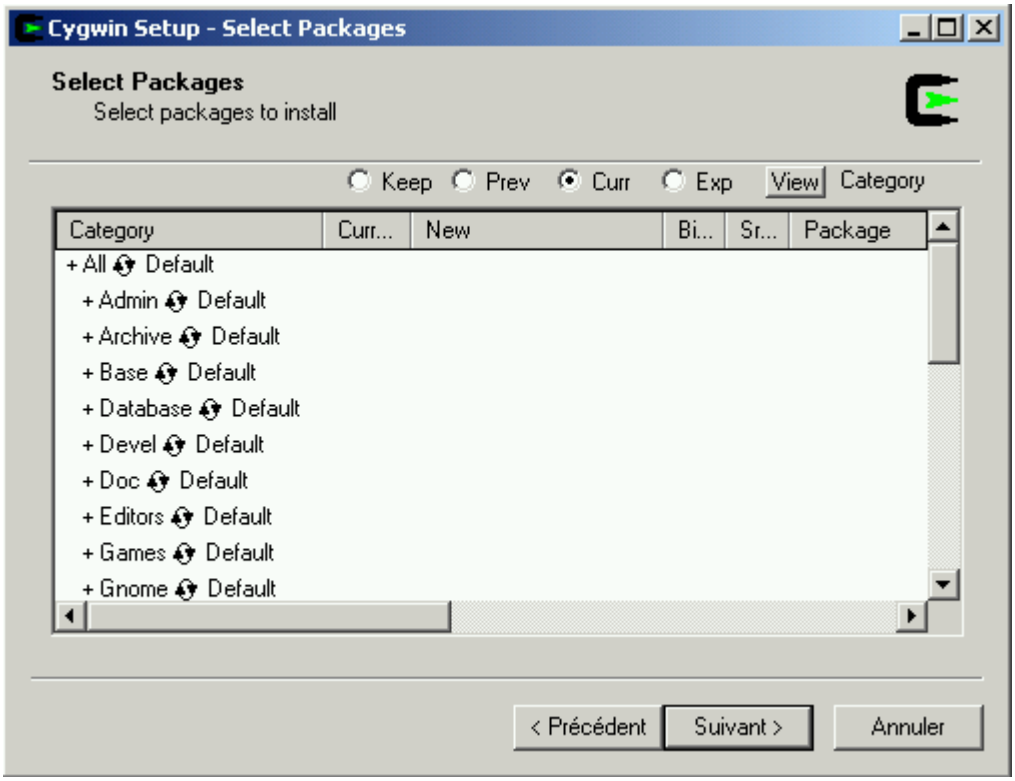

Figure 5: Liste des paquetages

Pour ce faire, il faut cliquer sur le mot *Default* situé sur la première ligne de la liste (commençant par *All*), à ce moment la, le programme affiche *Install* à la place de *Default* dans toute la liste, ce qui indique que tous les paquetages sont installés. Il faut noter que l'installation des tous les paquetages occupe plus de 2 Go sur le disque.

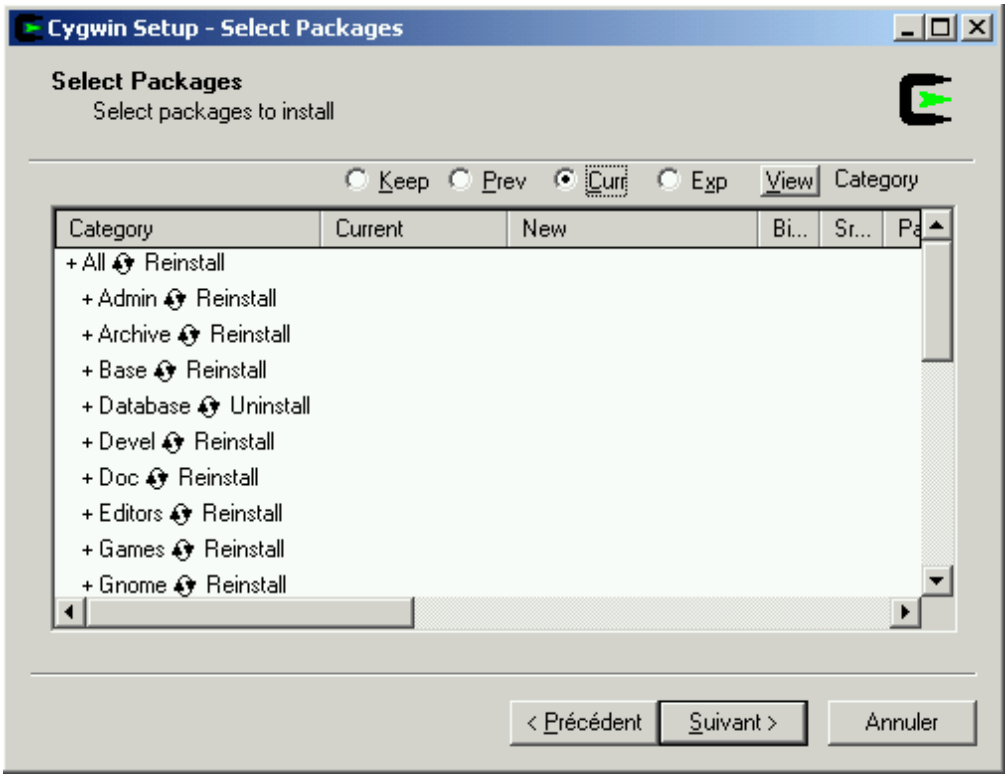

Figure 6: Sélection des paquetages

Si l'on clique sur *Suivant*, l'installation doit démarrer. En cas de blocage de l'installation à cause d'un problème d'accès réseau, il est possible d'interrompre le programme d'installation puis de l'exécuter de nouveau. L'installation reprendra au niveau du dernier paquetage installé.
Lorsque l'installation est terminée, on doit obtenir un icône *Cygwin* sur le bureau Windows, ce qui permet d'ouvrir le terminal CYGWIN qui a une fière allure de bon vieux terminal Linux.

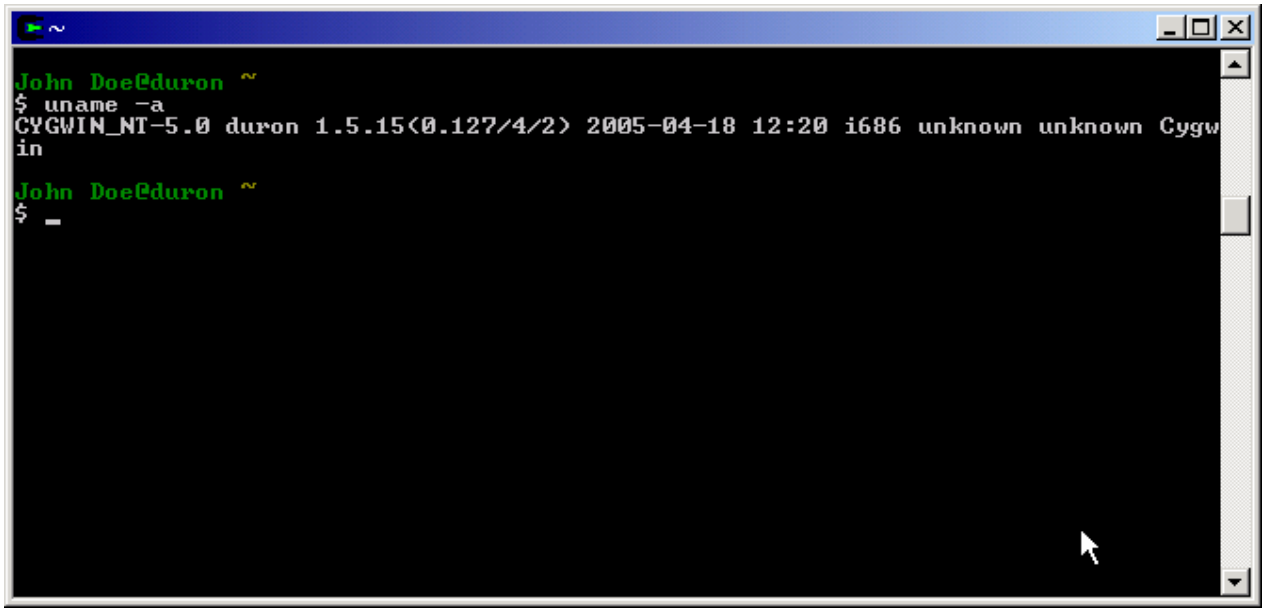

Figure 7: Terminal CYGWIN

En premier lieu, il faut exécuter la commande `rebaseall -v` dans le terminal. Cette commande est nécessaire car elle permet d'affecter une adresse de base unique aux différentes DLL et donc permettre un chargement ordonné des bibliothèques.
On peut alors démarrer l'environnement XFree86 qui par défaut affiche un émulateur de terminal `xterm`. A partir de ce terminal, on peut lancer les commandes Linux habituelles comme le montre la figure ci-dessous.

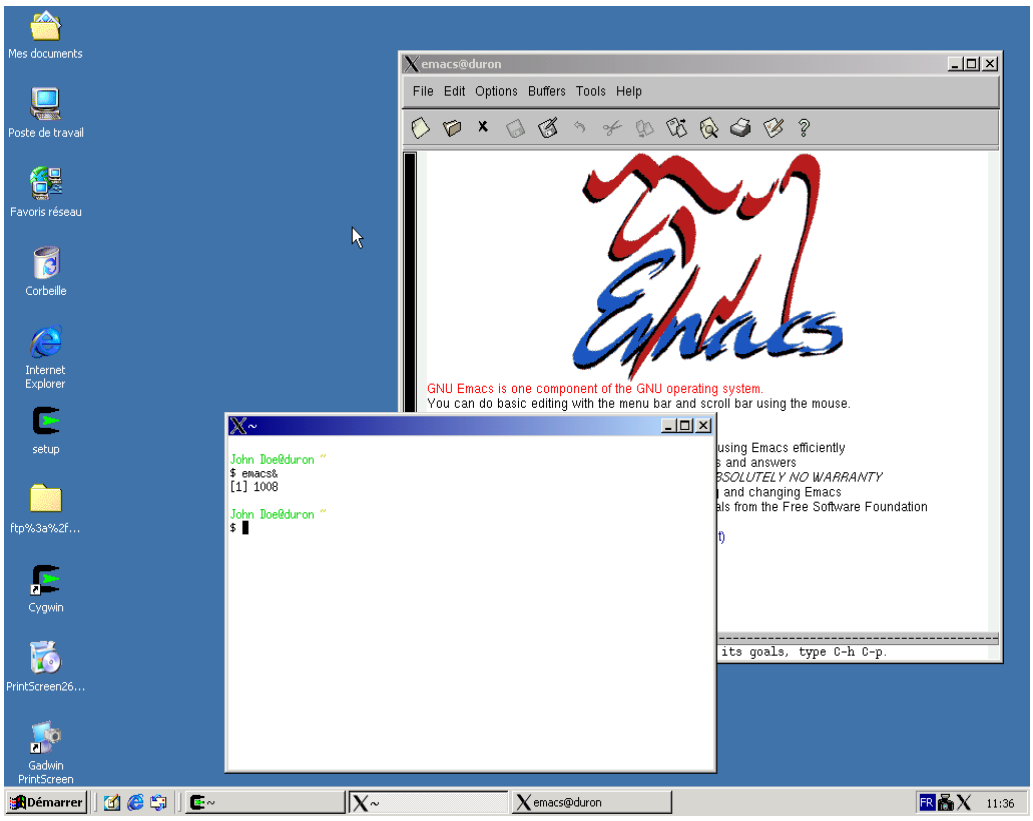

Figure 8: Environnement CYGWIN

Création de la chaîne de compilation croisée pour ARM

L'environnement Linux quasiment complet étant disponible, la procédure génération de la chaîne croisée CROSSTOOL est identique à celle utilisée sous Linux, décrite plus haut dans ce même chapitre. Il faut noter cependant que le type de plate forme de développement détecté par le script de configuration configure est dans notre cas *i686-host_pc-cygwin* au lieu de *i686-pc-linux-gnu*.

Attention:
Les scripts de CROSSTOOL ne s'entendent pas forcément très bien avec les noms de répertoire contenant des espaces, ce qui est fréquent sous Windows. Il est donc préférable d'extraire l'archive CROSSTOOL dans un répertoire ayant un nom UNIX comme `/home/test` et non pas `/home/mon répertoire`.

Après quelques heures de compilation, on obtient la même chaîne de développement incluant le compilateur `arm-linux-gcc`. On peut y accéder de la même manière que sous Linux.

```
$ export PATH=/opt/crosstool/arm/bin:$PATH
$ arm-linux-gcc -v
Reading specs from /opt/crosstool/arm/lib/gcc-lib/arm-linux/3.3.4/specs
Configured with: /home/test/crosstool-0.31/build/arm-linux/gcc-3.3.4-glibc-
2.3.2/gcc-3.3.4/configure --target=arm-linux --host=i686-host_pc-cygwin
--prefix=/opt/crosstool/arm --with-headers=/opt/crosstool/arm/arm-linux/include
--with-local-prefix=/opt/crosstool/arm/arm-linux --disable-nls --enable-
threads=posix --enable-symvers=gnu --enable-__cxa_atexit --enable-languages=c,c+
```

```
+ --enable-shared --enable-c99 --enable-long-long
Thread model: posix
gcc version 3.3.4
```

Et l'on peut bien sûr compiler l'exemple de la même manière.

```
$ pwd
/home/John Doe
$ arm-linux-gcc -o helloworld helloworld.c
$ file helloworld
helloworld: ELF 32-bit LSB executable, ARM, version 1 (ARM), for GNU/Linux
2.4.3, dynamically linked (uses shared libs), not stripped
```

Exemples de compilation croisée

Lorsqu'une chaîne de compilation est installée, il est possible de construire des outils à partir des paquetages. Voici quelques exemples.

Le noyau Linux pour ARM/AT91RM9200

L'architecture ARM est supportée par le noyau Linux mais il est nécessaire d'appliquer des patch disponibles sur http://www.arm-linux.org.uk/developer. Dans le cas d'un noyau 2.4.27, on doit tout d'abord extraire l'archive du noyau standard puis appliquer le patch générique ARM et enfin le patch spécifique au processeur ATMEL AT91RM9200.

```
# cd linux-2.4.27
# patch -p1 < 2.4.27-vrs1.patch
# patch -p1 < ../linux-patch/02_2.4.27-vrs1-at91-06102004.patch
```

La suite est très similaire à la compilation d'un noyau natif sauf que l'on doit utiliser les variables d'environnements `ARCH` et `CROSS_COMPILE` pour spécifier la chaîne de développement croisée . Pour configurer le noyau on utilisera la commande suivante:

```
# make ARCH=arm CROSS_COMPILE=arm-linux- menuconfig
```

Pour le compiler on utilisera la commande suivante.

```
# make ARCH=arm CROSS_COMPILE=arm-linux- dep zImage modules
```

Enfin pour installer les modules sur un répertoire image de la cible, on utilisera la commande suivante.

```
# make ARCH=arm CROSS_COMPILE=arm-linux- INSTALL_MOD_PATH=/target_arm
modules_install
```

Programme gdbserver

Ce programme permet de déboguer un programme exécuté sur la cible depuis le poste de développement à travers un lien réseau ou RS-232. Pour générer la version ARM on utilisera les commandes suivantes dans le répertoire des sources de `gdbserver`.

```
$ ./configure --host=arm-linux
$ make
```

Débogueur GDB croisé

Un tel outil permettra de déboguer à distance un programme exécuté sur une cible ARM via l'outil précédent.

```
$ ./configure –prefix=/opt/crosstool/arm --target=arm-linux --program-
prefix=arm-linux-
$ make
```

Débogueur GDB natif ARM

La commande suivante permet de construire une version de `gdb` directement utilisable sur la cible ARM.

```
$ ./configure --host=arm-linux –prefix=/usr/local/bin
$ make
```

Bibliothèque NCURSES native ARM

Nous construisons ici une version ARM de la bibliothèque NCURSES qui pourra être ajoutée à la chaîne de compilation.

```
$ BUILD_CC=gcc CC=arm-linux-gcc configure –host=arm-linux
--prefix=/opt/crosstool/arm/arm-linux --with-shared
$ make
```