Construire sous GNU/Linux des programmes pour Windows

Damien Goutte-Gattat <dgouttegattat@incenp.org>

2011/11/16

Résumé

Je présente dans ce document les différentes étapes permettant de construire, sans quitter son environnement GNU/Linux (mais ce serait, *a priori*, valable sur un autre Unix libre tel qu'un *BSD), des programmes ciblant Microsoft Windows.

Table des matières

1. Installer un compilateur croisé

Il semble que le premier réflexe de nombreux débutants voulant s'essayer à la cross-compilation sous GNU/Linux soit de rechercher (ou de demander sur un forum) « l'option à passer sur la ligne de commande de GCC pour lui demander de produire un .exe ». Coupons court à cela tout de suite : ça ne marche pas comme ça. Si vous utilisez le système *Truc* (par exemple GNU/Linux) sur une machine de type *Bidule* (par exemple x86), le compilateur dont vous disposez par défaut ne sera jamais capable de générer autre chose que des programmes ciblant le système Truc sur l'architecture Bidule. C'est normal, c'est ce que vous souhaitez la plupart du temps (quand vous n'avez pas besoin de cross-compiler) et ce n'est pas une simple option de ligne de commande qui va changer ça.

Pour pouvoir construire des exécutables ciblant un autre système que celui sur lequel vous travailler, il faut donc disposer d'un autre compilateur spécialement prévu à cet usage. De façon plus générale, pour chaque système pour lequel vous voulez générer des exécutables, vous devez avoir un compilateur dédié. Le choix de la cible se fait alors en invoquant le compilateur approprié, et non par une quelconque option de ligne de commande.

Donc, la première chose à faire est d'installer un compilateur croisé spécialement prévu pour générer des exécutables Windows (fichiers PE, *Portable Executable*). [1]

Selon la distribution GNU/Linux utilisée, il est possible qu'un tel compilateur croisé soit déjà disponible sous forme pré-compilée et empaquetée, n'attendant plus que d'être ins-

[1] Appréciez au passage le sens tout particulier que Microsoft donne au mot « portable » : chez eux, est portable tout ce qui fonctionne sur plus d'une version de Windows…

tallé via votre gestionnaire de paquets habituel. Dans ce cas, pas la peine de se casser la tête, faites confiance aux développeurs et mainteneurs de votre distribution. Le nom du ou des paquets variera probablement d'une distribution à l'autre, mais ça devrait ressembler à « mingw-*quelquechose* ».

Si la distribution ne fournit pas de compilateur croisé, il faudra le construire soi-même. Les sources et instructions nécessaires sont disponibles sur http://www.mingw.org/wiki/LinuxCrossMinGW, où se trouve entre autres un script permettant d'automatiser l'ensemble du processus.

Pour les utilisateurs de Slackware

Des slackbuilds [http://wiki.slackware-fr.org/slackbuilds:faq] pour Slackware-13.37 maintenus par votre serviteur sont disponibles auprès de la communauté Slackware francophone. Il vous suffit d'exécuter et d'installer dans l'ordre les slackbuilds d/binutils-mingw32 [http://sfo.svn.sourceforge.net/viewvc/sfo/slackbuilds-13.37/d/binutils-mingw32/], l/mingw32-libs [http://sfo.svn.sourceforge.net/viewvc/sfo/slackbuilds-13.37/l/mingw32-libs/] et d/gcc-mingw32 [http://sfo.svn.sourceforge.net/viewvc/sfo/slackbuilds-13.37/d/gcc-mingw32].

Une fois le compilateur croisé installé, vous pouvez passer à la section suivante. Le reste de cette section est destiné à ceux qui veulent installer un compilateur croisé « manuellement », ou du moins qui veulent savoir comment ça se passe.

1.1. Première étape : les binutils

Les binutils GNU [www.gnu.org/software/binutils/] sont un ensemble d'outils permettant de créer et de manipuler des fichiers exécutables. C'est un des principaux composants de la *toolchain* GNU.

Les binutils installés par défaut sur une distribution GNU/Linux manipulent des fichiers au format ELF (format exécutable standard des Unix récents). La première étape est donc de compiler une version des binutils destinée à manipuler des fichiers au format PE.

```
$ wget "http://downloads.sourceforge.net/mingw/binutils-2.20.1-src.tar.gz"
$ tar xf binutils-2.20.1-src.tar.gz
$ cd binutils-2.20.1-src
$ CFLAGS="-fno-exceptions"  \
  ./configure               \
  --prefix=/opt/mingw32     \
  --target=i686-pc-mingw32  \
  --with-gcc                \
  --with-gnu-as             \
  --with-gnu-ld             \
  --disable-nls             \
  --disable-debug           \
  --disable-shared
$ make
# make install
```

À propos des version

Les versions des paquetages utilisés ici correspondent à celles recommandées par le projet MinGW (lors de la rédaction de cette page). Je conseille de respecter ces recommandations. La cross-compilation est un domaine délicat et des versions plus récentes (donc moins testées) sont susceptibles de poser des problèmes.

En particulier, et tant que MinGW ne dira pas le contraire, il est fortement conseillé de s'en tenir à GCC 3.x et de ne pas chercher à tout prix à utiliser GCC 4.x.

Notez l'option `--target` passé au script `configure` : elle prend en argument un triplet de configuration [http://www.airs.com/ian/configure/configure_4.html] permettant d'identifier la plate-forme pour laquelle les binutils devront générer des exécutables — en l'occurence *i686-pc-mingw32*, soit un système Windows sur une machine de type PC à processeur Pentium Pro ou supérieur. Une même plate-forme peut être désignée par différents triplets, les *autotools* se chargeant de les résoudre en un triplet « canonique » non-ambigu. Par exemple, *i686-mingw32* est équivalent au précédent, la partie *manufacturer* étant simplement omise. On rencontre aussi fréquemment, toujours pour désigner la même plate-forme, des triplets du genre *i686-mingw32msvc*, mais rien ne justifie ce suffixe « -msvc » et son utilisation est déconseillée. [2]

L'exemple ci-dessus installe les binutils MinGW32 dans un dossier à part `/opt/mingw32` : tous les autres paquetages nécessaires seront installés au même endroit. Vous pouvez bien sûr l'adapter à votre guise (y compris en spécifiant un répertoire situé dans votre répertoire personnel, si vous ne voulez ou ne pouvez pas prendre les droits du super-utilisateur).

1.2. Deuxième étape : les fichiers d'en-tête

Le *runtime MinGW* est une bibliothèque statique permettant de générer des exécutables Windows « natifs » ne dépendant pas d'une DLL externe (telle que `cygwin.dll`). L'*API Windows*, quant à elle, fournit une interface vers l'ensemble des fonctionnalités de Windows.

Les fichiers d'en-tête de ces bibliothèques sont requis pour pouvoir compiler une première fois un compilateur croisé ciblant Windows.

```
$ wget "http://downloads.sourceforge.net/mingw32/mingwrt-3.18.tar.gz"
$ wget "http://downloads.sourceforge.net/mingw32/w32api-3.14.tar.gz"
$ tar xf mingwrt-3.18.tar.gz
$ tar xf w32api-3.14.tar.gz
# mkdir -p /opt/mingw32/include
# (cd /opt/mingw32 && ln -s . usr && ln -s . local)
# cp -r mingwrt-3.18/include /opt/mingw32
# cp -r w32api-3.14/include /opt/mingw32
```

Les fichiers d'en-tête sont donc copiés dans `/opt/mingw32/include`. Remarquez la création de deux liens `usr` et `local` dans `/opt/mingw32` pointant tous deux vers ce même répertoire : cela permet de simuler l'arborescence standard dans laquelle GCC s'attend à trouver les fichiers d'en-tête et de bibliothèque.

1.3. Troisième étape : GCC, première partie

Pour que tout fonctionne correctement, GCC doit être compilé deux fois : une première fois afin de pouvoir compiler les bibliothèques de MinGW32 et de l'API Windows (dont nous n'avons, pour l'instant, que les fichiers d'en-tête), et une seconde fois, *après* l'installation desdites bibliothèques.

La compilation de GCC nécessite les binutils préalablement installés, donc à partir de maintenant, il est nécessaire que le répertoire `/opt/mingw32/bin` figure dans le `PATH` ; ce sera également nécessaire une fois le cross-compilateur installé, pour pouvoir l'utiliser.

```
$ wget "http://downloads.sourceforge.net/mingw32/gcc-core-3.4.5-20060117-2-src.tar.gz"
$ tar xf gcc-core-3.4.5-20060117-2-src.tar.gz
$ cd gcc-3.4.5-20060117-2
$ mkdir build && cd build
$ CFLAGS="-fomit-frame-pointer"                \
  ../configure                                 \
```

[2] "Please ensure that [the configuration triplet] ends with *mingw32*, and *do not* emulate the asinine practice of GNU/Linux distributors, by appending *msvc*" — wiki MinGW [http://www.mingw.org/wiki/LinuxCrossMinGW]

```
  --prefix=/opt/mingw32                          \
  --target=i686-pc-mingw32                       \
  --enable-languages=c                           \
  --disable-debug                                \
  --disable-nls                                  \
  --disable-shared                               \
  --with-gcc                                     \
  --with-as=/opt/mingw32/bin/i686-pc-mingw32-as  \
  --with-ld=/opt/mingw32/bin/i686-pc-mingw32-ld  \
  --enable-sjlj-exceptions                       \
  --enable-threads=win32                         \
  --disable-win32-registry                       \
  --with-sys-root=/opt/mingw32
$ make
# make install
```

À noter, ici :

- l'option `--target`, qui a la même signification que précédemment pour les binutils ;
- l'option `--enable-languages=c`, qui indique de ne construire que le compilateur C (c'est le seul compilateur nécessaire pour compiler les bibliothèques de MinGW et de l'API Windows ; lors de la seconde compilation de GCC, il sera possible d'ajouter le support de langages supplémentaires, comme le C++ ou l'Objective C) ;
- l'option `--with-sys-root`, qui indique la racine de l'arborescence dans laquelle GCC ira chercher ses fichiers (il cherchera donc les fichiers d'en-tête dans `/opt/mingw32/usr/include`, les bibliothèques dans `/opt/mingw32/usr/lib`, etc.).

1.4. Quatrième étape : les bibliothèques de l'API Windows

Nous avons déjà installé les fichiers d'en-tête de l'API Windows, il faut maintenant compiler les archives contenant les fichiers objets (fichiers .a). Dans le répertoire d'extraction de `w32api3.14.tar.gz` :

```
$ mkdir build && cd build
$ CFLAGS="-mms-bitfields"   \
  ../configure              \
  --prefix=/opt/mingw32     \
  --host=i686-pc-mingw32    \
  --build=i686-pc-linux-gnu
$ make
# make install
```

Notez les options `--host` et `--build`. La première indique les fichiers générés seront *exécutés* sur la plate-forme *i686-pc-mingw32* (Windows, donc) ; il s'agit en effet de bibliothèques (statiques), donc même si elles sont installés sur le système GNU/Linux, elles seront en définitive incorporées à des exécutables Windows. La seconde option indique le système à partir duquel les bibliothèques sont compilées (*i686-pc-linux-gnu* soit un système GNU à noyau Linux sur une machine de type PC à processeur 32 bits Pentium Pro ou supérieur) ; cette option n'est en principe pas obligatoire (le script `configure` peut l'inférer lui-même), mais les développeurs des autotools recommandent de toujours préciser explicitement le système de build lorsqu'il est différent du système hôte.

1.5. Cinquième étape : le runtime MinGW

Même chose que ci-dessus. Depuis les sources de MinGW :

```
$ mkdir build && cd build
$ CFLAGS="-mms-bitfields"   \
  ../configure              \
  --prefix=/opt/mingw32     \
  --host=i686-pc-mingw32    \
```

```
  --build=i686-pc-linux-gnu
$ make
# make install
```

1.6. Dernière étape : GCC, seconde partie

Les fichiers d'en-tête et les bibliothèques installés, il ne reste plus qu'à compiler une seconde fois GCC. Cette fois-ci, outre le compilateur C, on peut y ajouter le support d'autres langages, comme le C++, l'Objective C, le Java, etc. Le code nécessaire est empaqueté dans des archives distinctes de celle du cœur de GCC, il suffit d'extraire le contenu de ces archives par-dessus.

Depuis les sources de GCC :

```
$ mkdir build2 && cd build2
$ CFLAGS="-fomit-frame-pointer" \
  ../configure                  \
  --prefix=/opt/mingw32         \
  --target=i686-pc-mingw32      \
  --enable-languages=c,c++      \
  --disable-debug               \
  --disable-nls                 \
  --disable-shared              \
  --with-gcc                    \
  --with-gnu-as                 \
  --with-gnu-ld                 \
  --enable-sjlj-exceptions      \
  --enable-threads=win32        \
  --disable-win32-registry      \
  --with-sys-root=/opt/mingw32
$ make
# make install
```

Pour faire plus propre, il est possible de supprimer les répertoires `doc`, `info` et `man` qui ont été créés dans `/opt/mingw32`, et qui ne contiennent rien d'utile (puisque la documentation qu'ils contiennent est déjà disponible dans `/usr`).

2. Essai de compilation manuelle

Maintenant qu'un compilateur croisé est installé, testez que tout fonctionne comme prévu en compilant un programme simple (un classique *Hello, world!* fait parfaitement l'affaire). Il suffit d'utiliser **i686-pc-mingw32-gcc** au lieu de simplement **gcc** ; adaptez éventuellement le préfixe « i686-pc-mingw32 » pour qu'il corresponde au triplet de configuration utilisé lors de la compilation du compilateur croisé. Et bien sûr, assurez-vous que le répertoire contenant le compilateur croisé et les outils associés (`/opt/mingw32/bin` si vous avez suivi la procédure ci-avant) est dans le `PATH`.

```
$ i686-pc-mingw32-gcc hello.c
$ ls
a.exe hello.c
$ file a.exe
a.exe: MS-DOS executable PE  for MS Windows (console)
$ ./a.exe
-bash: ./a.exe: No such file or directory
$ wine a.exe
Hello, world!
```

Nous pouvons voir ici que le compilateur croisé à généré un fichier `a.exe` (et non `a.out` comme l'aurait fait le compilateur natif), que ce fichier est bien identifié par **file** comme un exécutable Windows, et qu'une tentative de l'exécuter directement sous GNU/Linux se solde par une erreur (pas très explicite à première vue d'ailleurs). En revanche, si on exécute le programme par l'intérmediaire de Wine [http://www.winehq.org/], on obtient bien le résultat attendu.

3. Compiler un projet autoconfisqué

Un projet « autoconfisqué », ou *autoconfiscated*, est un projet utilisant le système de build GNU, généré par l'ensemble d'outils appelé les *autotools* — dont l'un des plus importants est autoconf [http://www.gnu.org/software/autoconf/]. Concrètement, et en première approximation, dès lors que les sources d'un projet contiennent un script `configure`, c'est un projet autoconfisqué. [3]

Le système de build GNU supporte d'office la compilation croisée. Dès lors qu'un compilateur croisé est disponible, il suffit d'invoquer le script `configure` avec l'option `--host` pour indiquer que l'on souhaite compiler pour un autre système que le système courant.

Le fait que le système de build supporte la compilation croisée ne signifie pas pour autant que *tous* les projets utilisant ce système seront cross-compilables. Le *code* lui-même doit être portable, soit en évitant tout ce qui est spécifique à un système donné, soit en proposant des alternatives en fonction du système. À noter que les autotools peuvent aider à écrire du code portable.

Illustrons la compilation croisée d'un projet autoconfisqué avec l'exemple de la bibliothèque libiconv [http://www.gnu.org/software/libiconv/] :

```
$ tar xf libiconv-1.12.tar.bz2
$ cd libiconv-1.12
$ ./configure               \
  --prefix=/opt/mingw32     \
  --build=i686-pc-linux-gnu \
  --host=i686-pc-mingw32    \
  --disable-nls             \
  --with-gnu-ld
$ make
# make install
```

L'option `--prefix` installera la bibliothèque cross-compilée dans le répertoire `/opt/mingw32`, où a déjà été installé le compilateur croisé si vous avez procédé à une installation « manuelle » en suivant la procédure ci-dessus ; ce n'est pas spécialement nécessaire, la bibliothèque peut très bien être installée n'importe où ailleurs. Je l'installe ici pour deux raisons : a) pour que le compilateur croisé puisse trouver tout seul les fichiers d'en-tête et les fichiers objets (pas besoin des options `-I` et `-L`), b) parce que je trouve simplement logique de rassembler tous les logiciels et bibliothèques relatifs à la compilation croisée au même endroit.

Les options `--build` et `--host` indiquent respectivement le système sur lequel la compilation a lieu (donc le système courant) et le système cible des binaires à générer.

Regardons les fichiers générés par la compilation et l'installation :

```
/opt/mingw32/include/libcharset.h
/opt/mingw32/include/iconv.h
/opt/mingw32/include/localecharset.h
```

Aucune surprise ici, ce sont les fichiers d'en-tête de la bibliothèque.

```
/opt/mingw32/lib/libiconv.la
/opt/mingw32/lib/libiconv.dll.a
/opt/mingw32/lib/libcharset.la
/opt/mingw32/lib/libcharset.dll.a
```

Les fichiers `.*a` sont les fichiers objets, qui seront nécessaires lors de la phase d'édition des liens. Les fichiers `*.la` sont nécessaires au fonctionnement de Libtool [http://www.gnu.org/software/libtool/], une composant des autotools facilitant la création de bibliothèques dynamiques.

[3] Ce n'est pas strictement vrai, le script `configure` peut être généré par d'autres outils voire même être écrit « à la main » par le développeur — mais l'utilisation d'autoconf reste le cas le plus fréquent.

```
/opt/mingw32/bin/iconv.exe
/opt/mingw32/bin/libiconv-2.dll
/opt/mingw32/bin/libcharset-1.dll
```

Finalement, voici l'utilitaire de conversion d'encodage **iconv** (ici naturellement appelé `iconv.eve`, s'agissant d'un exécutable Windows) et les deux bibliothèques dynamiques. Seuls ces deux derniers fichiers seront nécessaires pour exécuter sous Windows un programme lié à ces bibliothèques ; tous les autres fichiers ne sont requis qu'à la compilation.

4. Créer le programme d'installation

Une fois le programme cross-compilé, il reste à le distribuer à ses utilisateurs. La méthode la plus simple est de rassembler les binaires et les fichiers associés dans une archive (au format ZIP de préférence, plus commun sous Windows que le TAR), mais ce n'est pas la façon habituelle de procéder pour les programmes Windows. L'usage veut plutôt que l'on fournisse à l'utilisateur un *programme d'installation* qui se chargera de toutes les étapes nécessaires à l'installation (et à la désinstallation).

Le programme d'installation peut lui aussi être généré depuis GNU/Linux. Il existe au moins un générateur de programmes d'installation utilisable sur ce système : NSIS [http://nsis.sourceforge.net/].

Comme pour le compilateur croisé, installez-le via votre gestionnaire de paquets habituel s'il est disponible dans les dépôts de votre distribution. Sinon, compilez les sources comme suit (le compilateur croisé doit être préalablement installé) :

```
$ wget "http://downloads.sourceforge.net/nsis/nsis-2.46-src.tar.bz2"
$ tar xf nsis-2.46-src.tar.bz2
$ cd nsis-2.46
# scons install
```

NSIS est un programme 32 bits uniquement. Si vous utilisez un système 64 bits, vous devez avoir un compilateur permettant de générer des binaires 32 bits (option `-m32` de **gcc**). Pour les utilisateurs de Slackware64, cela implique d'installer les paquets multilib [http://alien.slackbook.org/dokuwiki/doku.php?id=slackware:multilib] de Eric Hameleers.

Une fois installé, NSIS peut être invoqué par la commande **makensis**. L'utilisation de NSIS, et en particulier le langage de script qu'il utilise pour décrire les programmes d'installation et de désinstallation, sort du cadre de ce document, je vous renvoie donc à sa documentation [http://nsis.sourceforge.net/Docs/Contents.html].