Développement natif - Démonstration 2 : développer un programme *natif* en C, fiche de suivi

Objectifs

- Comprendre la phase de *linkage* avec des librairies (fournie par la plateforme ou les nôtres)
- Créer et distribuer sa propre librairie partagée (*shared library*)

Situation initiale

> On parlera indistinctement de *library*, librairie ou bibliothèque. On parle souvent de librairie en français malgré le fait que la traduction correcte de library soit bibliothèque. Néanmoins, le mot librairie est plus court et c'est un faux ami qui permet de faire la correspondance du concept dans les deux langues. Et puis, une librairie et une bibliothèque proposent toutes les deux des livres à utiliser donc le sens n'est pas totalement perdu.

Nous avons une *library*, du code, `mylib` que nous souhaiterions distribuer. Cette library contient des fonctions et des structures de données *utiles*. Si nous voulons distribuer ce code, nous n'allons distribuer que le header (`mylib.h`) et le binaire, et non le code source de l'implémentation (`mylib.c`), car l'utilisateur n'a pas à la connaître pour s'en servir. Il a seulement besoin de connaître les *déclarations* (l'API de ma library).

Pour cela, je vais donc distribuer aux utilisateur·ices :

- Le header `mylib.h`, pour que l'utilisateur connaisse les signatures des fonctions
- Mon implémentation compilée sous forme de librairie partagée (*shared library*) (`.so`)

Qu’est ce qu’une librairie partagée (shared library) ?

Une librairie partagée est un *fichier objet* (binaire, *code source compilé*). Sous Unix, les shared libraries sont appelées *shared object* (d’où l’extension `.so`), sous Windows elles sont appelées *dynamic link libraries* (ou DLLs, d’où l’extension `.dll`). Cela permet de partager des implémentations sous forme de binaire : fonctions, structures de données, etc..

Le code ainsi compilé peut être utilisé *par plusieurs programmes en même temps* (d’où le *shared library*). La librairie partagée est *linkée* de manière dynamique au *run-time* (chargée en mémoire une fois, au moment de l’exécution).

Créer une librairie partagée

Pour transformer `mylib` en code distribuable, on crée une librairie partagée (ou *shared library*) `libmylib.so`

```
#Compilation (vers une plateforme cible)
gcc -c mylib.c -o mylib.o
#Création d'une librairie dynamique en liant des fichiers objets (ici un seul)
gcc -shared mylib.o -o libmylib.so
```

La librairie partagée `libmylib.so` est crée. Je distribue `mylib.h` et `libmylib.so` aux utilisateurs. Ma librairie partagée *est spécifique à une plateforme* (c’est du code compilé !). Si elle est compilée pour GNU/Linux, elle ne pourra être éxecutée par Windows ou Android par exemple.

Utiliser la librairie dynamique dans son projet

En tant qu’utilisateur, je récupère une copie de ces deux fichiers (`mylib.h` et `libmylib.so`) pour les utiliser dans mon propre projet (ici `main.c`)

Compiler

1. Pour pouvoir utiliser la librairie mylib dans mon projet, quelle est la première chose à faire ?

```
#include "mylib.h"
```

Je compile mon projet, cela me crée un *object file* `myapp.o`

```
gcc -o myapp.o -c main.c
```

Linker

Le binaire `main.o` n’est pas encore executable car il contient des *références* vers des libraires : `mylib` (`struct Foo`, `createFoo`, `destroyFoo`) et `stdio.h` (`printf`). Inclure le header a permis

de fournir des déclarations, donc lors de la phase de compilation, le compilateur a seulement vérifier que les fonctions et structures existaient et avaient la bonne signature.

Le fichier objet contient encore ses références, elles doivent à présent être remplacées par le code source (du binaire) pour fabriquer l'executable. C'est l'étape de linkage (*linking*).

2. Ouvrir le fichier `myapp.o`. Que remarque-t-on ? Que reste-il à faire pour produire l'executable ?

   Le système a des moyens (il est configuré pour) pour trouver tout seul où se trouve la librairie compilée de `stdio.h` (le binaire s'appelle `libc.so`). Mon executable sera linké de manière dynamique par le linker au binaire de `printf` (qu'il sait trouver) qui sera chargé en mémoire.

Il faut donc ici seulement linker le fichier objet `main.o` avec la librairie dynamique `libmylib.so`

```
#Création de l'exécutable main avec linkage dynamique vers la librairie.
        Plusieurs options
##Option 1
gcc -L$(pwd) main.o -lmylib -o myapp -Wl,-rpath,$(pwd)
##Option 2 (s'assurer qu'/usr/local/lib est sur LD_LIBRARY_PATH)
sudo cp libmylib.so /usr/local/lib/
gcc -L$(pwd) main.o -lmylib -o myapp
```

Executer

```
./myapp
```

3. **Écrire** un `Makefile` pour automatiser la compilation de la librairie partagée `mylib` et la compilation de mon programme `myapp` qui utilise cette librairie.

   Pour écrire un Makefile, commencez par vous demander quelle est la cible (fichier à construire), de quoi dépend-il et enfin comment le construire.

   En situation réelle, on séparerait ces deux phases de build dans deux Makefile séparés. Chaque projet (la librairie et l'application) aurait son Makefile et son *dépôt*.

4. Exécutez deux fois d'affilée la règle pour construire la librairie. Que remarquez-vous ?

5. Modifier le code source de la librairie et ajouter une fonction `substract` qui calcule la différence entre deux structures `Foo`. Ré-executer le programme `myapp`.
6. Supposons que je souhaite utiliser `mylib` (`mylib.h` et `mylib.so`) sur Windows. Puis-je l'utiliser telle quelle ? Pourquoi ? Quelles solutions s'offrent à cet utilisateur pour s'en servir ?

Conclusion de cette démo

- Le linkage consiste à remplacer les références vers des déclarations de fonctions/variables par leur code binaire

- Le linkage est une étape cruciale pour fabriquer un executable. Il permet d'utiliser les librairies fournies par la plateforme (cela fait partie de son SDK)
- Le processus de linkage est spécifique à chaque OS
- Chaque OS a sa propre API (binary format, system calls et fonctions, etc.)
- Pour executer un programme issu d'un même code source vers une autre plateforme (par exemple de Linux à Windows), il *faut le recompiler* (ou utiliser une couche logicielle d'émulation comme WINE ou Cygwin) et avoir une partie du SDK en commun