TP2 - Les Bibliothèques sous UNIX

Introduction au projet Nibbler

David Pineau pineau_d@epitech.net
Maxime Montinet zaz@epitech.net
David Giron thor@epitech.net

Résumé: Le Nibbler est un jeu video à l'image de snake, où l'on vous demande d'être capables de lancer votre jeu avec une GUI différente à la demande. Pour répondre à cette problématique, vous utiliserez des bibliothèques dynamiques manipulées à l'exécution. L'utilisation de vos bibliothèques dynamiques dans votre programme principal devra être générique. Le but de ce TP est donc de vous familiariser avec ce mecanisme et de vous donner les premières pièces pour réussir votre projet.

Table des matières

I Retour sur ld.so

I.1 Exercice 1

- Lisez la page de manuel de `dlopen(3)`, `dlsym(3)`, `dlclose(3)` et `dlerror(3)`.

I.2 Exercice 2

Nous profiterons de ce retour sur `ld.so` pour introduire les notions d'initialisation et de finalisation de vos bibliothèques dynamiques.

Au chargement et au déchargement d'une bibliothèque dynamique, il est possible d'appeler une fonction de votre choix pour initialiser ou finaliser votre bibliothèque. Pour cela, on utilise deux attributs de `gcc` :

- `__attribute__((constructor))`
- `__attribute__((destructor))`

La fonction portant l'attribut `constructor` est appelée à la fin de l'exécution de `dlopen` avant le retour de cette fonction. La fonction portant l'attribut `destructor` fonctionne de la même facon, mais est appelée à la fin de l'exécution de `dlclose` avant le retour de cette fonction.

- Vous allez créer trois bibliothèques partagées, aux noms à votre convenance en n'utilisant que `gcc`.
- Chacune de ces bibliothèques devra contenir 3 fonctions
  - Une procédure portant l'attribut `constructor` et ne prenant pas de paramètres.
  - Une procédure portant l'attribut `destructor` et ne prenant pas de paramètres.
  - Une fonction au prototype de votre choix nommée `entry_point`.
- Chaque fonction devra afficher sur la sortie standard un message pertinent lié à son rôle.
- A l'aide d'une boucle, chargez chaque bibliothèque, appelez la fonction `entry_point` et fermez chaque bibliothèque.
- Votre programme principal ne doit faire aucun affichage !

Indices `Au contraire d'une fonction, une procédure ne revoie pas de valeur.`

Votre affichage pourra ressembler à cela :

```
[libga] Loading library...
[libbu] Loading library...
[libzo] Loading library...
[libga] Entry point...
[libbu] Entry point...
[libzo] Entry point...
[libga] Unloading library...
[libbu] Unloading library...
[libzo] Unloading library...
```

Indices Afin de pouvoir utiliser les fonctions *dlopen(3)*, *dlsym(3)* et *dlclose(3)*, vous devez linker votre programme avec la bibliothèque libdl et donc par conséquent utiliser l'option -ldl.

II Passage au C++

II.1 Rappel

Comme vous l'avez vu en cours, il existe en C++ un mécanisme particulier de la compilation permettant le polymorphisme paramétrique. Ce mécanisme est la `décoration de symboles`, aussi appelé `mangling`, et il consiste à encoder le nom des fonctions avec le type de leurs paramètres ainsi que les namespaces dans lesquels elles sont contenues.

Cela a malheureusement pour effet de générer des symboles aux noms obscurs...

Comme vous l'aurez probablement compris, la chaîne à passer en paramètre à la fonction `dlsym(3)` doit correspondre au nom exact du symbole. Mais comment faire si le symbole est modifié au mangling et qu'on ne connait pas ce nom ?

En C++, il existe une solution pour remédier à ce problème : La syntaxe `extern "C"` :

```
1   extern "C"
2   {
3     int entrypoint(void)
4     {
5   // Code du point d'entree de la bibliotheque
6     }
7   }
8
9   // Ou encore...
10  extern "C" int entrypoint(void)
11  {
12    // Code ici...
13  }
```

II.2 Exercice 3

Modifiez le code de vos 3 bibliothèques dynamiques de l'exercice précédent pour que vous puissiez les compiler en utilisant `g++`.

II.3 Avec des objets

La methode la plus simple en C++ afin de créer une interface core-bibliothèque qui soit un minimum `objet` est donc de proposer un point d'entrée qui servira en réalité de créateur d'objets, comme vu en cours.

Pour ce faire, votre programme possèdera la déclaration d'une interface (le fichier `.h`), qui sera implementée par une classe de la bibliothèque. Le point d'entrée peut alors renvoyer une instance de cette classe (en la manipulant via l'interface en question), que le programme principal peut alors utiliser.

Indices `Référez-vous au cours si cette partie vous semble encore floue.`

Partons de l'interface suivante pour la suite des exercices :

```
1   class IDisplayModule
2   {
3     public:
4     virtual ~IDisplayModule() {}
5
6     virtual std::string const & getName() const = 0;
7
8     virtual void init() = 0;
9     virtual void stop() = 0;
10  };
```

II.4 Exercice 4

- Modifiez vos 3 bibliothèques dynamiques pour qu'elles possèdent chacune une classe au nom de votre choix implémentant l'interface `IDisplayModule`.
- Chacune des méthodes `init()` et `stop()` affichera un message sur la sortie standard différant d'une bibliothèque a l'autre.
- Assurez-vous que chacune de vos bibliothèques possède un point d'entrée générique permettant de récupérer une instance polymorphique de la classe contenue implémentant l'interface.

III Chargeur de bibliothèques

Les fonction proposées par la bibliothèque `libdl` correspondent à une API C. Comme nous sommes en C++, nous n'allons bien entendu pas utiliser *directement* les fonctions dlopen, dlsym, et dlclose.

La solution que nous vous proposons est d'encapsuler ces fonctions dans un objet dont le rôle sera de charger une bibliothèque et de récupérer des instances d'un objet depuis celle-ci.

III.1 Exercice 5

- Complétez la classe `DLLoader` suivante :

```
1  class DLLoader
2  {
3    public:
4    [...]
5    IDisplayModule* getInstance([...]);
6    [...]
7  };
```

- La fonction getInstance devra renvoyer une instance de la classe contenue dans votre bibliothèque.
- Le type DLLoader sera instancié une fois par bibliothèque à charger.

Indices Vous pouvez rajouter autant de fonctions/attributs que vous le désirez, mais gardez à l'esprit qu'un objet simple est un objet fiable ! Ne mélangez pas les métiers !!!

IV Un vrai chargeur de bibliothèques

Votre `DLLoader` est bien sympathique, mais il manque encore un petit (!) quelque chose. Actuellement, vous pouvez récuperer depuis une bibliothèque dynamique n'importe quel objet qui implémente `IDisplayModule`. Seulement, quand vous aurez besoin d'une autre interface, vous allez devoir refaire un autre DLLoader pour cette interface, et ainsi de suite...

Vous allez donc améliorer votre `DLLoader` en faisant en sorte qu'il puisse récupérer un objet répondant à une interface au choix, avec un point d'entrée dans la bibliotheque au choix. Ainsi, il sera compatible avec (presque) toutes les bibliothèques C++ que vous réaliserez.

La solution que nous vous proposons d'implémenter est un `DLLoader` template sur le type de l'interface à prendre en compte.

IV.1 Exercice 6

- Modifiez votre classe `DLLoader` de la façon suivante :

```
1  template <typename T>
2  class DLLoader
3  {
4    public:
5    [...]
6    T* getInstance([...]);
7    [...]
8  };
```

Indices Pourquoi ne pas en profiter pour abstraire également le nom du point d'entrée dans la bibliothèque ?

IV.2 Exercice 7

Utilisez le temps qu'il vous reste pour aller plus loin ! Pourquoi ne pas envisager un gestionnaire de bibliothèques dynamiques utilisant votre `DLLoader` avec un système de types réflechi et des fonctionnalités étendues ? Souvenez-vous : le bon développeur est fainéant. Il développe des outils une bonne fois pour toutes !

Soyez imaginatifs, et bon courage !