TP3 - Bibliothèques partagées

Création de systèmes modulaires

Koalab koala@epitech.eu

Abstract: Durant ce TP, vous allez apprendre a utiliser des bibliothèques dynamiques sous Windows. Vous réaliserez également une abstraction au système d'exploitation appliquée aux bibliothèques dynamiques, et poserez les premières pierres d'un système de gestion de plug-ins générique.

Table des matières

Chapitre I

Préambule au TP

I.1 Cadre et esprit de ce TP

Vous avez dejà appris à utiliser des bibliothèques dynamiques sous UNIX en tech2. Ce TP a plusieurs objectifs :

- Vous faire utiliser des bibliothèques sous Windows.
- Vous faire réaliser une abstraction à l'OS appliquée aux bibliothèques dynamiques.
- Vous apprendre à utiliser les bibliothèques dans un contexte de plug-ins, à savoir la gestion de plusieurs bibliothèques, chargées en même temps, apportant chacune un outil à votre programme.

Comme d'habitude, il n'est pas acceptable de faire le TP à moitié, car *TOUTES* les notions appliquées dans ce TP seront par la suite considérées comme *ACQUISES*, et leur application sera exigée en **soutenance** !

Ne cédez pas à la flemme !

I.2 Remarques diverses

Nous attachons une grande importance à la qualité de nos documents. Si vous constatez dans ce sujet des erreurs, des fautes d'orthographe ou de typographie, ou si vous avez des remarques quelconques à faire, contactez-nous à koala@epitech.eu.

I.3 Pré-requis

Afin de faire ce TP, vous allez avoir besoin des choses suivantes :

- Un Windows et un UNIX de votre choix (Windows 7 / Fedora 13 dump étudiant ou similaires)

- Un Visual Studio en état de fonctionnement
- Un éditeur UML (Violet UML Editor, graphviz, PowerAMC, boUML, Dia, ...)
- Un cerveau en bon état !

Chapitre II

Bibliothèques dynamiques sous Windows

II.1 Un peu de lecture

Faites vous les dents sur la section de la MSDN relative aux bibliothèques dynamiques : http://msdn.microsoft.com/en-us/library/ms682599%28v=VS.85%29.aspx

Prêtez une attention particulière aux fonctions *LoadLibrary*, *GetProcAddess* et *FreeLibrary*, qui sont les équivalents des fonctions POSIX *dlopen*, *dlsym* et *dlclose*, et qui vous serviront à la même chose !

II.2 Rappels

Comme vous l'avez déjà vu en tech 2, afin de charger un symbole depuis une bibliothèque compilée en C++, vous devez utiliser la syntaxe `extern "C"` :

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

Dans les exercices de ce TP, vous devrez récupérer depuis le programme appelant un type interne à la bibliothèque. Vous **DEVEZ** utiliser un système de point d'entrée unique en *extern "C"* renvoyant un pointeur sur une interface implementée par le type en question.

II.3 Exercice 1 (Windows)

- Réalisez deux bibliothèques dynamiques, contenant les types *Platypus* et *Armadillo*.
- Ces types **DOIVENT** implémenter une interface de votre choix qui leur permettra (au moins) de *crier(void)*.

Les cris doivent être réalistes ! Le premier qui fait miauler un *Platypus* fera l'objet d'une subtile reprogrammation à l'aide d'un Outil de Reprogrammation Non-Volatile

- Réalisez un main qui ouvre vos deux bibliothèques, récupère les objets contenus dedans, et les fait crier.
- Faites valider votre travail par un assistant.

Vous devrez bien évidemment utiliser les fonctions Windows de manipulation de bibliothèques dynamique, vous n'avez pas le droit à la lib *dl*...

C'était facile, n'est-ce pas ?

Chapitre III

Abstraction à l'OS

III.1 Exercice 2 (Windows)

Améliorez votre exercice 1 en utilisant un objet chargeur de bibliothèque, comme vous avez fait en tech2 :

- Créez un objet *DlLoader* contenant au moins la fonction membre suivante :
  - *IVotreInterface* getInstance(void)*
- Votre *DlLoader* ne **DOIT PAS** exposer le type du handle de la bibliothèque, et doit donc encapsuler le handle.
- Modifiez votre main pour utiliser votre chargeur de bibliothèque et montrer son bon fonctionnement.

`Afin que votre système soit réutilisable, il serait intelligent de le templater ...`

III.2 Exercice 3 (Windows + UNIX)

Maintenant que vous possédez votre chargeur de bibliothèques sous Windows, il est temps de réaliser votre abstraction à l'OS appliquée aux bibliothèques dynamiques qui vous servira pour le restant de l'année.

- Réalisez une interface au chargeur de bibliothèque que vous avez précedemment réalisé. Appelez la *IDlLoader* ou autre nom explicite.
- Réalisez une implémentation sous UNIX de cette interface.
- Modifiez votre main afin de montrer que votre code compile et s'éxécute de la même façon sous UNIX et Windows.

- Faites valider votre travail par un assistant.

Cette abstraction est le **minimum syndical** à utiliser dans vos projets !

Chapitre IV

Plugin manager

IV.1 Préambule

On peut distinguer deux utilisations des bibliothèques dynamiques :

- Ouverture d'une seule bibliothèque à la fois, par exemple pour déterminer un comportement du programme au lancement (Choix d'une interface graphique par exemple).
- Ouverture de plusieurs bibliothèques en même temps, pour réaliser un système de plug-ins ou de modules.

Vous avez déjà couvert le premier aspect en tech2. Il est temps maintenant de s'attaquer au deuxième aspect.

Pour cela, vous allez avoir besoin d'un système de gestion des "modules" ouverts, que vous allez réaliser dans l'exercice suivant.

IV.2 Exercice 4 (Windows + UNIX)

- Vous devez concevoir un système de gestion de "modules".
- Réfléchissez à un type *DLManager*, qui devra obéir aux contraintes suivantes :
  - Utilise votre *IDlLoader* pour charger les bibliothèques.
  - Permet d'ouvrir un nombre virtuellement infini de bibliothèques.
  - Permet d'ouvrir toutes les bibliothèques contenues dans un répertoire donné (Par exemple *./plugins* ...)
  - Permet d'identifier les bibliothèques par une *std : :string* représentant leur "type" (Par exemple *platypus* pour celle contenant l'objet *Platypus*, ...)
  - Permet de récupérer un objet contenu dans l'une des bibliothèques ouvertes.

- Votre *DlManager* devra également etre templaté sur l'interface implementée par les objets à récupérer depuis les bibliothèques.

Vous **DEVEZ** concevoir votre *DlManager* en UML et le faire valider par un assistant avant de le réaliser !

- Une fois votre conception finalisée, vous pouvez implémenter votre *DlManager* en C++.
- Testez votre *DlManager* en l'utilisant pour le code des exercices précédents, sous Windows et UNIX.