TP Système

Gestionnaire de plugins

Introduction

Un plugin est un composant logiciel qui permet d'ajouter dynamiquement une fonctionnalité à un logiciel, au cours de son exécution.

Le but de ce TP est d'écrire un petit système de gestion de plugins permettant d'ajouter dynamiquement des filtres de traitement d'image à une mini-application ; ce TP vous fera découvrir :

1. les pointeurs de fonction ;
2. les bibliothèques chargées dynamiquement : ouverture, recherche puis exécution d'une fonction ;
3. les plugins ;
4. la bibliothèque de traitement d'image *OpenCV*.

1 Analyse

Pointeur de fonction

1. Compilez (avec la commande make), exécutez puis analysez les deux fichiers sources ptrfn_v1.c et ptrfn_v2.c pour comprendre :
   — la notion de pointeur de fonction,
   — la syntaxe de la déclaration d'un tel pointeur,
   — l'utilisation d'un pointeur de fonction,
   — l'utilisation et l'intérêt de typedef ;
2. Pour comprendre le Makefile fourni, ne pas hésiter à consulter la page `http://perso.univ-rennes1.fr/jean-christophe.engel/make`

Charger dynamiquement une bibliothèque

1. Compilez (avec la commande make), exécutez puis analysez le fichier dynload_v1.c pour comprendre :
   — le mécanisme d'ouverture d'une bibliothèque dynamique,
   — la recherche puis l'exécution d'une fonction ;

   Consultez la documentation (man dlopen) pour en savoir plus.
   — quel fichier d'entête faut-il inclure pour la compilation ?
   — quelle bibliothèque faut-il ajouter pour l'édition de liens ?
2. Faites une version interactive (dynload_v2.c) de ce programme qui permet à l'utilisateur de saisir le nom d'une fonction (à un seul paramètre) de la bibliothèque mathématique, qui trouve cette fonction puis qui l'exécute (exemples de fonctions : sqrt, cos, exp, ...).
   N'hésitez pas à consulter la documentation de la bibliothèque mathématique : man math.h.

Fabriquer une bibliothèque dynamique (appelée aussi bibliothèque partagée)

1. Mettez les trois fonctions de calcul (plus, moins, mult) de la première partie dans un fichier calcul.c ;
   — vérifiez, dans la documentaion du compilateur (man gcc), la signification de l'option -fPIC ; contrôlez son utilisation dans le Makefile.
   — vérifiez, dans la documentaion du compilateur (man gcc), la signification de l'option -shared ; contrôlez son utilisation dans le Makefile.
   — utilisez la commande make pour fabriquer la bibliothèque libcalcul.so.
   — affichez avec la commande readelf –dyn-syms libcalcul.so la table des symboles dynamiques, et vérifiez la présence des trois fonctions de calcul.
2. Combinez les programmes ptrfn_v2.c et dynload_v2.c dans un fichier main_calcul.c pour effectuer les traitements suivants :
   — ouvrir la bibliothèque libcalcul.so ;

   - effectuer la saisie de 2 nombres entiers ;
   - afficher un menu avec les trois opérations de la bibliothèque de calcul et demander à l'utilisateur quelle opération effectuer ;
   - chercher la fonction correspondante dans la bibliothèque, l'appliquer sur les 2 nombres saisis et afficher le résultat avec la fonction afficher définie précédemment.
3. Complétez le Makefile pour produire l'exécutable main_calcul à partir de main_calcul.c.
4. Exécutez le programme main_calcul ; vous devriez obtenir une erreur indiquant que la bibliothèque libcalcul.so n'est pas trouvée. En relisant la documentation (man dlopen), notez comment dlopen cherche une bibliothèque lors de son ouverture. Dans notre cas, nous utiliserons la variable LD_LIBRARY_PATH pour indiquer au chargeur où trouver la bbliothèque : faites le nécessaire pour que votre programme puisse charger la bibliothèque.
5. Faites une deuxième version de main_calcul.c :
   - dans la fonction afficher :
     - supprimez le paramètre calculer puis ajoutez un troisième paramètre entier ; la fonction doit se contenter d'afficher ses 4 paramètres ;
     - placez le prototype de cette fonction dans un fichier afficher.h à inclure dans ce fichier source ;
   - dans le main, effectuez directement l'appel de la fonction de calcul, sans passer par la fonction afficher ;
6. Faites une deuxième version de calcul.c :
   - chaque fonction de calcul doit appeler la fonction afficher *avant* de renvoyer son résultat ; le dernier paramètre de afficher doit être le résultat de l'opération programmée dans la fonction appliquée aux deux autres ;
   - ne pas oublier d'inclure afficher.h.
7. Complétez le Makefile pour fabriquer :
   - une deuxième bibliothèque (il faudra d'ailleurs changer son nom dans la deuxième version de main_calcul)
   - une deuxième version de main_calcul ;
8. Exécutez le programme main_calcul_v2 ; vous devriez obtenir une nouvelle erreur... cette fois-ci, c'est la fonction afficher qui n'est pas trouvée lorsqu'une des fonctions de la bibliothèque dynamique s'exécute ;
   - utilisez readelf pour afficher la table des symboles dynamiques de main_calcul_v2 : on constate que la fonction afficher n'y figure pas et c'est la cause du problème précédent.
   - vérifiez dans la documentation de l'éditeur de liens (man ld), la signification de l'option -export-dynamic ; complétez le Makefile, recompilez et relancez l'exécution : le problème devrait être réglé.

2 Notion de plugin

Pour qu'une application puisse être étendue dynamiquement par l'ajout de plugin, elle doit disposer d'un système de gestion de plugins, qui incorpore les mécanismes suivants :

découverte de plugins : mécanisme qui pemet en cours d'exécution de se rendre compte de l'installation de nouveaux plugins dans un endroit (répertoire) prédéfini ;

enregistrement de plugin : mécanisme par lequel un plugin s'inscrit dans une application pour lui ajouter une nouvelle fonctionnalité, c'est-à-dire rendre une ou plusieurs nouvelle(s) fonction(s) disponible(s) dans l'application ; le gestionnaire doit mémoriser les différentes fonctions de chaque plugin, de façon à pouvoir les appeler à la demande de l'application ; il faut en outre que ces fonctions respectent une interface *normalisée*.

Programmation de plugins en C : En C, un plugin se réalise à l'aide de bibliothèques dynamiques : le mécanisme de chargement dynamique permet d'ajouter et d'exécuter de nouvelles fonctions en cours d'exécution.

- le gestionnaire de plugin cherche les bibliothèques dynamiques dans un ou plusieurs répertoires puis les charge ;
- chaque plugin possède une fonction de nom prédéfini (par exemple : init_) qui peut être appelée par le gestionnaire : cette fonction sert à enregistrer dans le gestionnaire une ou plusieurs fonctions de traitement fournies par le plugin ;
- l'application peut ensuite appeler ces nouvelles fonctions (à l'aide du gestionnaire) et les fonctions du plugin peuvent appeler des fonctions de l'application.

OpenCV

L'application que nous voulons programmer utilise la bibliothèque de traitement d'image *OpenCV* (`http://opencv.org`) ; chaque plugin devra permettre de réaliser un traitement simple sur une image, par appel d'une primitive de la bibliothèque *OpenCV* :

- lire une image depuis un fichier
- enregistrer une image dans un fichier
- afficher une image
- appliquer un traitement sur une image (filtre)
- ...

La documentation en ligne vous permettra d'en savoir davantage ; il est tout particulièrement utile de lire certains des tutoriels d'introduction :

- comment lire et afficher une image : `http://docs.opencv.org/doc/tutorials/introduction/display_image/display_image.html`
- comment lire, modifier, afficher et enregistrer une image : `http://docs.opencv.org/doc/tutorials/introduction/load_save_image/load_save_image.html`

Cette bibliothèque fournit une API C++... que nous utiliserons depuis un système programmé en C : voir plus loin.

Image

Une image est une matrice de points (pixels) ; chaque pixel représente une valeur de luminosité. Pour simplifier une image est représentée par deux sortes d'informations :

- des meta-données qui décrivent les caractéristiques de l'image (hauteur, largeur, nombre de bits par pixel, etc...)
- les pixels

Le type cv : :Mat permet de représenter des images dans le système *OpenCV* ; il gère les deux sortes d'informations ci-dessus, ainsi que l'allocation et la libération de la mémoire requise pour les pixels.

Filtre

Un filtre est un petit traitement appliqué à une image ; dans le cas de notre application, chaque filtre sera programmé à l'aide d'une fonction qui prend une image en paramètre et rend une image en résultat ; cette fonction sera placénse dans un plugin destiné à être chargé dynamiquement dans l'application. Par conséquent, la signature de chaque fonction de filtre de notre système sera : cv::Mat filtre (cv::Mat src)

Gestionnaire de plugin

Le fichier pluginmanager.h contient des déclarations de types et de fonctions destinés à être utilisés par l'application et par les plugins, qui devront donc l'importer (directive #include) ; ce fichier vous est donné et ne devra pas être modifié : il représente l'interface publique du gestionnaire de plugins.

On y définit les types suivants :

- filter_function : type d'une fonction de filtrage

```
  // fonction de réalisation du filtre
typedef
cv::Mat (* filter_function)(cv::Mat src);
```

- plugin_descriptor : informations mémorisées par le gestionnaire pour chaque plugin

```
// descripteur de plugin
typedef
struct {
  const char *     m_name;                 // nom du filtre
  const char *     m_description;          // description de l'effet du filtre
  filter_function  m_filtre;               // fonction de traitement du filtre
} plugin_descriptor;
```

— plugin_manager : c'est le type du gestionnaire de plugin ; sa définition est masquée et devra être faite dans le fichier pluginmanager.cc.

```
// struct plugin_manager_t est le type du gestionnaire
// c'est une structure dont le contenu est à définir dans pluginmanager.cc
// et qui n'est pas dévoilée
struct plugin_manager_t;

// plugin_manager est synonyme de struct plugin_manager_t
typedef struct plugin_manager_t plugin_manager;
```

Ce fichier définit aussi la signature de plusieurs fonctions publiques du gestionnaire ; il faudra les implémenter dans le fichier pluginmanager.cc.

Certaines de ces fonctions sont destinées à être utilisées par l'application :

```
// initialiser un manager
plugin_manager * make_manager();

// libérer les ressources du gestionnaire
void release_manager(plugin_manager * pm);

// découvrir les plugins dans un répertoire
unsigned int discover_plugins(const char dirname[], plugin_manager * pm);

// chercher et renvoyer le plugin de numéro donné
plugin_descriptor * get_plugin(plugin_manager * pm, unsigned int plugin_number);

// afficher le menu des plugins disponibles
void   display_menu(plugin_manager * pm);
```

La fonction register_plugin ci-dessous doit être utilisée par un plugin pour enregistrer la fonction de filtrage auprès du gestionnaire, lors de la découverte du plugin par le gestionnaire ; le gestionnaire n'enregistre la fonction que si elle n'est pas déjà présente.

```
// enregistrer un plugin auprès du gestionnaire
void
register_plugin(plugin_manager * pm,
                const char filter_name[],
                const char filter_description[],
                filter_function the_filter);
```

Structures de données internes du gestionnaire de plugin Le gestionnaire va conserver l'ensemble des descripteurs de plugin dans une liste simplement chaînée :

— cette liste sera initialisée dans la fonction make_manager() ;
— lors de la découverte d'un plugin (fonction discover_plugins()), son descripteur sera ajouté dans la liste, *s'il n'y est pas déjà* ; la fonction qui réalise cette opération (register_plugin()) devra être appelée par la fonction d'initialisation du plugin, qui est elle-même appelée par le gestionnaire ;
— la fonction get_plugin() parcourt la liste des plugins pour trouver le plugin demandé et renvoyer son descripteur ;
— la liste est vidée par la fonction release_manager().

Étapes

à programmer dans plugin_manager.cc :

1. Définir la structure de la liste chaînée ;
2. Définir le type struct plugin_manager_t : cette structure contient les informations gérées par le gestionnaire de plugins ;
3. Programmer les fonctions make_manager(), register_plugin(), get_plugin(), release_manager() et display_menu() ; cette dernière fonction devra afficher un titre, le numéro de chaque plugin suivi de sa description, ainsi qu'un choix pour mettre fin à l'application.
4. Programmer la fonction discover_plugins() ; pour cette fonction vous utiliserez les fonctions :
   — opendir : « ouvre » un répertoire, alloue et renvoie un descripteur (DIR *) à utiliser avec readdir ;
   — closedir : « ferme » le répertoire et libère le descripteur alloué par opendir ;
   — readdir (man **-s 3** readdir) : permet de parcourir les entrées du répertoire ouvert par opendir à l'aide d'un descripteur (DIR *) ;

   La fonction devra parcourir les entrées du répertoire paramètre, sélectionner les fichiers contenant des plugins (*_plugin.so, constante plugin_suffix), charger la bibliothèque, y chercher la fonction d'initialisation du plugin (constante initfunc_name), appeler cette fonction (qui se chargera d'appeler register_plugin).

   Vous aurez besoin d'utiliser quelques fonctions de la bibliothèque de chaînes C : man string.

Vous veillerez à sécuriser au maximum votre programme par un traitement judicieux des différents cas d'erreur.

Programme applicatif

Dans un autre fichier vous programmerez une fonction main qui :
— détermine le répertoire où sont situés les plugins, soit par argument d'appel, soit par saisie ;
— permet à un utilisateur de sélectionner interactivement un filtre puis l'applique sur une image ;

Compilation, édition de liens

L'application, le gestionnaire de plugins et les plugins sont programmés en C, mais le tout doit être compilé par le compilateur C++.

Vous aurez remarqué que le contenu des différents fichiers est encapsulé dans la directive extern « C » { ... } ;

Utilisez la commande make pour fabriquer le programme imagesystem ; ce programme doit pouvoir s'exécuter sans erreur.

Utiliser un premier plugin.

Dans le répertoire plugins se trouve un premier exemple de plugin très basique : fabriquez la bibliothèque dynamique correspondante puis exécutez à nouveau le programme imagesystem : il devrait découvrir le plugin ci-dessus et vous devriez pouvoir l'exécuter sans erreur.

Programmer des plugins

Sur le modèle du plugin fourni, fabriquez les plugins qui permettent de réaliser les traitements ci-dessous, à raison d'un traitement par plugin ; les tutoriels référencés au début de ce document vous donnent toutes les informations nécessaires à la programmation des différents traitement demandés.
— lire une image ; la saisie du nom du fichier qui contient l'image sera effectuée par le plugin ; plusieurs images sont disponibles dans le répertoire image ;
— afficher une image ;
— appliquer un flou (*blur*) sur une image ;

Votre système devrait permettre, en cours d'exécution, de découvrir et d'exécuter les nouveaux plugins installés dans le répertoire de plugins.

Autres plugins (facultatifs)

— enregistrer une image ; le plugin effectue la saisie du nom du fichier ; l'extension (png, jpg, tiff, ...) définit le format d'enregistrement de l'image ;
— tout autre fonction simple de modification de l'image compatible avec le fonctionnement du système.