
Module SE1 — ENIB

Labo — 3h — F. Harrouet

Bibliothèques dynamiques

But du Labo

- ▷ Réaliser et utiliser une bibliothèque dynamique.
- ▷ Mettre en évidence le principe de chargement des symboles.
- ▷ Réaliser une application à base de *plugins*.

1 Une bibliothèque triviale

Nous disposons du programme `sincos.c` qui se contente d'afficher le *sinus* et le *cosinus* de la valeur passée en ligne de commande. Il utilise les fonctions `sin()` et `cos()` déclarées dans `math.h` et est par conséquent lié à la bibliothèque mathématique `libm.so` (option `-lm` dans le `makefile`).

Après avoir essayé ce programme, il faudra compléter le fichier `antiTrigo.c` afin de fournir nos propres fonctions `sin()` et `cos()` ; ces fonctions se contenteront de renvoyer une valeur quelconque. Ce fichier sert à fabriquer la bibliothèque `libAntiTrigo.so`. Pour l'utiliser il faut modifier le `makefile` afin de lier le programme à notre bibliothèque à la place de la bibliothèque mathématique. L'exécution du programme `sincos` doit désormais afficher les valeurs fournies par nos propres fonctions `sin()` et `cos()` :

```
{user@host}$ ./sincos 0.3
sin(0.3)=0.3          cos(0.3)=0.3
```

2 Préchargement de symboles

Il faut maintenant restaurer le `makefile` afin que le programme `sincos` soit à nouveau lié à la bibliothèque mathématique (`-lm`) et non plus à `libAntiTrigo.so`. Le programme doit retrouver un comportement correct :

```
{user@host}$ ./sincos 0.3
sin(0.3)=0.29552     cos(0.3)=0.955336
```

Nous utiliserons cette fois le *préchargement* de symboles afin de détourner le programme de son fonctionnement initial sans le modifier :

```
{user@host}$ LD_PRELOAD=./libAntiTrigo.so ./sincos 0.3
sin(0.3)=0.3          cos(0.3)=0.3
```

À ce stade, nous sommes donc capables d'intercepter les appels d'un programme vers des fonctions de bibliothèque. Nous venons de fournir du code qui se substitue complètement aux fonctionnalités originales mais nous pouvons aller plus loin en utilisant malgré tout ces fonctionnalités.

Nous devons pour cela compléter la fonction `initPointers` du fichier `antiTrigo.c` pour qu'elle retrouve les symboles `sin` et `cos` dans la bibliothèque `libm.so`. Nos propres

versions de ces fonctions se contenteront alors d'invoquer la fonction d'initialisation si nécessaire (le premier appel) et d'invoquer les versions originales pour renvoyer l'opposé du résultat :

```
{user@host}$ LD_PRELOAD=./libAntiTrigo.so ./sincos 0.3
sin(0.3)=-0.29552     cos(0.3)=-0.955336
```

Ceci devient intéressant lorsqu'on utilise des programmes que nous ne maîtrisons pas :

```
▷ echo 'plot [-3.14:3.14] sin(x), cos(x)' | \
  ( LD_PRELOAD=./libAntiTrigo.so gnuplot -persist )
▷ LD_PRELOAD=./libAntiTrigo.so xeyes
```

Comparer l'exécution avec/sans le préchargement.

Les pages de manuel suivantes contiennent la documentation nécessaire :

```
▷ ld.so(8) dlopen(3)
```

3 Application extensible par *plugins*

Il s'agit de réaliser une calculatrice reposant sur l'utilisation d'une pile de valeurs. L'utilisateur est invité à saisir des valeurs réelles, qui sont alors empilées, ou des noms d'opérateurs qui sont appliqués. La calculatrice ne connaît aucun opérateur *a priori* et utilise le chargement dynamique pour importer des *plugins* qui effectuent les diverses opérations. Ainsi cette calculatrice peut être enrichie et modifiée (au niveau de ces opérateurs) sans être recompilée ni même arrêtée !

Dans le répertoire `Calc` se trouve le programme `calc.c` avec la fonction `calc_run()` à compléter. Il est accompagné de `calc.h` qui sera utilisé par les divers opérateurs. Dans le sous-répertoire `Oper` il faudra réaliser les divers *plugins*. Chacun d'eux devra naturellement fournir une fonction correspondant au type `CalcOperation` (défini dans `calc.h`) et dont le nom sera choisi de manière cohérente avec l'opération de chargement réalisée dans `calc_run()`.

Tester les propriétés dynamiques du procédé :

- ▷ un opérateur manque ; il est fabriqué dans un autre terminal et devient disponible,
- ▷ un opérateur fonctionne d'une certaine façon ; il est modifié dans un autre terminal et se comporte alors différemment,

et tout ceci sans arrêter le fonctionnement de la calculatrice.

S'il reste du temps :

- ▷ Comment faire pour que les *plugins* fournissent un objet *C++* devant jouer un rôle dans l'application (à la place d'une simple fonction) ?
- ▷ Cet objet doit bien entendu être d'un type dérivé d'un type connu (et utilisé) par l'application principale.
- ▷ ex : l'application connaît la classe abstraite `Operator` et l'utilise à travers sa méthode virtuelle pure `eval()` ; chaque *plugin* permet de créer un objet dérivé d'`Operator` dont la méthode `eval()` effectue le traitement utile.

```
#include <math.h>
#include <stdio.h>

int
main(int argc,
     char ** argv)
{
    double d;
    if((argc!=2) || (sscanf(argv[1], "%lg", &d)!=1))
    {
        fprintf(stderr, "usage: %s value\n", argv[0]);
        return 1;
    }
    fprintf(stdout, "sin(%g)=%g\tcos(%g)=%g\n", d, sin(d), d, cos(d));
    return 0;
}
```

```
#include <math.h>
#include <dlfcn.h>
#include <string.h>
#include <stdio.h>

double (*ptr_sin)(double);
double (*ptr_cos)(double);

void
initPointers(void)
{
    /* ... A completer ... */
}

double
sin(double d)
{
    /* ... A completer ... */
}

double
cos(double d)
{
    /* ... A completer ... */
}
```

```
/*-----*/  
#include "calc.h"  
  
int  
main(void)  
{  
    Calc *calc=calc_new();  
    calc_run(calc);  
    calc_delete(calc);  
    return 0;  
}  
/*-----*/
```

```
/*-----*/  
#ifndef CALC_H  
#define CALC_H 1  
  
#include <stdio.h>  
  
#define OPER_DIRECTORY "./Oper"  
  
typedef struct _Calc Calc;  
  
typedef void (*CalcOperation)(Calc *);  
  
Calc *  
calc_new(void);  
  
void  
calc_delete(Calc *calc);  
  
size_t  
calc_stackSize(const Calc *calc);  
  
void  
calc_push(Calc *calc,  
           double value);  
  
double  
calc_pop(Calc *calc);  
  
void  
calc_showStack(const Calc *calc,  
               FILE *stream);  
  
void  
calc_run(Calc *calc);  
  
void  
calc_stop(Calc *calc);  
  
#endif /* CALC_H */  
/*-----*/
```

```

/*-----*/
#include "calch"

#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <readline/readline.h>
#include <readline/history.h>

#include <sys/types.h>
#include <dirent.h>

#include <dlfcn.h>

struct _Calc
{
int stop;
size_t capacity;
size_t size;
double *stack;
};

/*-----*/

Calc *
calc_new(void)
{
Calc *calc=(Calc *)malloc(sizeof(Calc));
calc->stop=0;
calc->capacity=8;
calc->size=0;
calc->stack=(double *)malloc(calc->capacity*sizeof(double));
return calc;
}

void
calc_delete(Calc *calc)
{
free(calc->stack);
free(calc);
}

size_t
calc_stackSize(const Calc *calc)
{
return calc->size;
}

void
calc_push(Calc *calc,
double value)
{
if(calc->size==calc->capacity)
{
calc->stack=(double *)realloc(calc->stack,
(calc->capacity<<=1)*sizeof(double));
}
calc->stack[calc->size++]=value;
}

double
calc_pop(Calc *calc)
{

```

```

return calc->stack[--(calc->size)];
}

void
calc_showStack(const Calc *calc,
FILE *stream)
{
size_t i;
for(i=0;i<calc->size;++i)
{
if(i)
{
fputc(' ',stream);
}
fprintf(stream,"%g",calc->stack[i]);
}
}

static
char *
getCompletion(const char *text,
int state)
{
static char *candidates[0x100];
static int nbCandidates;
if(!state)
{
size_t len=strlen(text);
DIR *d=opendir(OPEP_DIRECTORY);
nbCandidates=0;
if(d)
{
struct dirent *e=readdir(d);
while(e)
{
char buffer[0x100];
if(sscanf(e->d_name,"calc%s",buffer)==1)
{
char *ext=strstr(buffer,".so");
if(ext)
{
*ext='\0';
if(!strncmp(text,buffer,len))
{
candidates[nbCandidates]=malloc(strlen(buffer)+1);
strcpy(candidates[nbCandidates++],buffer);
}
}
}
e=readdir(d);
}
closedir(d);
}
return state<nbCandidates ? candidates[state] : (char *)0;
}

static
char *
getline(const char *prompt)
{
char *line=readline(prompt);
if(line)
{

```

```

size_t len=strlen(line);
if(len)
{
    char *begin;
    char *end;
    for(begin=line;*begin;++begin)
        if(!isspace(*begin))
            break;
    for(end=line+len-1;end>begin;--end)
        if(!isspace(*end))
            break;
    len=end+1-begin;
    if(line!=begin)
        memmove(line,begin,len);
    line[len]='\0';
}
if(len)
    add_history(line);
else
{
    free(line);
    line=(char *)0;
}
return line;
}

void
calc_run(Calc *calc)
{
    rl_completion_entry_function=&getCompletion;
    rl_bind_key('\t',rl_complete);
    calc->stop=0;
    while(!calc->stop&&!feof(stdin))
    {
        char *line=getLine("? ");
        if(line)
        {
            double value;
            if(sscanf(line,"%lg",&value)==1)
            {
                calc_push(calc,value);
            }
            else
            {
                char libName[0x100];
                snprintf(libName,0x100,"%s/calc%s.so",OPER_DIRECTORY,line);
                /**
                 * ... A COMPLETER ...
                 */
            }
        }
        free(line);
    }
}

```

```

    }
    fputs("stack: ",stdout);
    calc_showStack(calc,stdout);
    fputc('\n',stdout);
}

void
calc_stop(Calc *calc)
{
    calc->stop=1;
}

/*-----*/

```