Complément de cours : librairies statiques et dynamiques

Arash Habibi & Frédéric Vivien

Ce document porte sur les librairies, leur utilisation et la manière de les générer. Pour la réalisation du TP, vous pouvez passer directement à la section 2. La section 1 porte sur la nature des fichiers objet.

1 Les fichiers objet

Les premières étapes de la compilation, c'est-à-dire toutes celles qui se situent avant celle de l'édition de liens, consistent à transformer les fichiers dits *sources* (extension *.c*) en fichiers dits *objet* (extension *.o*). Prenons par exemple le fichier source `alloctabint.c` suivant :

```
#include <stdlib.h>
int* alloctabint(int nb)
{
   int *tab;
   tab = (int*)malloc(nb*sizeof(int));
   if(tab==NULL)
   {  perror("malloc"); exit(1); }
   return(tab);
}
```

La commande

```
gcc -c alloctabint.c -o alloctabint.o
```

génère un fichier `alloctabint.o` qui est un fichier objet. Les fichiers objet contiennent des informations codées en binaire. Ces informations représentent le programme contenu dans le fichier source dans un format presque exécutable. En particulier :

- si le fichier source contient des appels à des fonctions qui ne sont pas définies dans ce fichier (dans notre exemple les fonctions `malloc`, `exit`, et `perror`) alors ces appels sont représentés par des branchements à des adresses mémoires restant à déterminer ultérieurement ;
- si le fichier source contient des déclarations de variables `extern`, alors l'adresse mémoire de ces variables reste également indéterminée.

Par ailleurs, la seule chose qui pourrait manquer à un fichier objet pour être un fichier exécutable à part entière, est une fonction `main`.

Étant donné un fichier objet contenant une fonction `main`, et des fichiers objet où seraient définies les fonctions `exit`, `malloc` et `perror`, l'éditeur de liens calcule l'adresse mémoire de ces fonctions et les met à la disposition de la fonction `alloctabint` qui pourra alors faire des branchements vers ces fonctions. L'éditeur de liens calcule aussi l'adresse mémoire de la fonction `taballocint`. S'il existe des fonctions qui appellent `alloctabint`, l'éditeur de lien leur fournira l'adresse de cette fonction ce qui leur permettra de faire des branchements vers elle. Enfin, après l'édition de liens, le processeur aura accès à l'adresse de la fonction `main` ce qui lui permettra de lancer l'exécution du programme.

2 Les librairies

2.1 Description

Les librairies contiennent la définition de fonctions susceptibles d'être utilisées par plusieurs programmes (simultanément ou non). Il existe deux types de librairies : les librairies *statiques* et les librairies *dynamiques*. Le nom des librairies statiques est typiquement de la forme :

`lib***.a`

Par exemple en regardant dans le répertoire `/usr/lib` vous trouverez :

- `libc.a` est la librairie standard C (fonctions `malloc`, `exit`, etc.) ;
- `libm.a` est la librairie mathématique (fonctions `sqrt`, `cos`, etc.) ;
- `libjpeg.a` est la librairie des fonctions permettant de manipuler les fichiers image au format jpeg ;
- `libGL.a` est la librairie des fonctions permettant de manipuler des objets 3D ;
- etc.

Le nom des librairies dynamiques est typiquement de la forme :

`lib***.so`

Dans le répertoire `/usr/lib` vous trouverez également les versions dynamiques des librairies ci-dessus.

- Utiliser des librairies statiques, revient à inclure la définition des fonctions de la librairie dans votre fichier exécutable, pendant l'étape de l'édition de liens (donc pendant la compilation et *avant* le lancement du programme).
- Utiliser des librairies dynamiques, revient à indiquer à votre programme l'emplacement d'où il pourra charger en mémoire ces définitions *après* le lancement du programme.

L'avantage des librairies statiques est que le fichier exécutable qui en résulte contient, avant l'exécution, tout ce qui lui est nécessaire pour fonctionner. Alors que, si une librairie dynamique a disparu, ou a été considérablement modifiée, un programme exécutable qui s'exécutait parfaitement en utilisant cette librairie peut devenir totalement inopérant.

Par contre, un programme obtenu par compilation avec une librairie statique a un fichier exécutable beaucoup plus volumineux que le même programme obtenu par compilation avec une librairie dynamique, puisque la définition des fonctions de la librairie ne se trouve pas dans le fichier exécutable.

Enfin, si une librairie statique est mise à jour alors, tout programme l'utilisant devra être recompilé pour qu'il puisse prendre en compte la modification. Dans le cas d'une librairie dynamique, cette mise à jour n'a pas besoin de recompilation.

2.2 Utilisation

2.2.1 Principe général

Soit la librairie `libXXX.a` (ou `libXXX.so`) se trouvant dans un répertoire dont le chemin absolu est `chemin`. Pour compiler un fichier source `prog.c` faisant appel à des fonctions de cette librairie, il faut taper la ligne de commande suivante :

```
gcc prog.c -Lchemin -lXXX -o prog
```

autrement dit, après l'option `-l`, il faut mettre le nom de la librairie sans l'extension (donc `.a` , `.so`) et sans le le préfixe `lib`.

- Pour la librairie `libjpeg.a`, l'option de compilation est `-ljpeg` ;
- pour la librairie `libGL.so`, l'option de compilation est `-lGL` ;
- pour la librairie `libsocket.so.2`, l'option de compilation est `-lsocket` ;
- et ainsi de suite.

Si le programme contient plusieurs fichiers source : `prog1.c`, `prog2.c`, `prog3.c`, ..., `progn.c`, prog`main.c`, alors il faudra la suite de commandes suivante :

```
gcc -c prog1.c -o prog1.o
gcc -c prog2.c -o prog2.o
gcc -c prog3.c -o prog3.o
  ...
gcc -c progn.c -o progn.o
gcc -c main.c -o main.o
gcc prog1.o prog2.o prog3.o ... progn.o main.o -Lchemin -lXXX -o prog
```

On peut aussi simplement exécuter :

```
gcc prog1.c prog2.c prog3.c ... progn.c main.c -Lchemin -lXXX -o prog
```

2.2.2 Exemple

Soit le programme suivant qui calcule et affiche la racine carrée du nombre qu'on lui fournit en entrée. On utilise pour cela, la fonction `sqrt` qui a été défini dans le fichier `libm.a`, mais aussi dans le fichier `libm.so` qui se trouvent tous deux dans le répertoire `/usr/lib`. Voici le fichier `prog.c` :

```
#include <math.h>
int main()
{
    double in;
    scanf("%f",&in);
    printf("%f\n", sqrt(in));
    return 0;
}
```

Pour compiler ce programme, il faut exécuter la commande :

```
gcc prog.c -L/usr/lib -lm -o prog
```

2.2.3 Différence d'utilisation des librairies statiques et dynamiques

Dans ce qui précède, nous n'avons fait aucune distinction entre les librairies statiques et dynamiques. En effet l'utilisation des deux types de librairies est presque identique. Il y a pourtant une petite différence : dans le cas des librairies dynamiques, si le programme allait toujours chercher les librairies au même emplacement, il suffirait de changer cet emplacement pour que le programme devienne inutilisable [1], ou qu'il faille le recompiler. C'est pourquoi pour chercher l'emplacement des librairies dynamiques, on s'aide d'une variable d'environnement appelée

```
LD_LIBRARY_PATH
```

Cette variable indique au programme à quels emplacements il doit chercher les librairies dynamiques. Si cet emplacement est modifié, il suffit de modifier la variable, sans changer le programme. Pour indiquer au système qu'il faut chercher dans le répertoire `/usr/local/lib`, il faudra initialiser la variable `LD_LIBRARY_PATH` de la manière suivante :

```
export LD_LIBRARY_PATH=/usr/local/lib
```

Si l'on veut que les programmes cherchent dans `/usr/local/lib`, dans `/usr/X11R6/lib` et dans le répertoire courant, il faudra écrire :

```
export LD_LIBRARY_PATH=.:/usr/X11R6/lib:/usr/local/lib
```

En pratique, vous ne définirez *jamais* cette variable mais vous ajouterez des fichiers à sa définition :

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/users/profs/habibi/libs
```

2.3 Création de librairies

Soient les fichiers `prog1.c`, `prog2.c`, `prog3.c`, ..., `progn.c` contenant des fonctions (autres que `main`). Nous voulons mettre ces fonctions dans une librairie pour que d'autres programmes puissent les utiliser. Dans un premier temps, il est nécessaire de compiler ces fichiers pour obtenir des fichiers objet (cf. section 1).

```
gcc -c prog1.c -o prog1.o
gcc -c prog2.c -o prog2.o
gcc -c prog3.c -o prog3.o
   ...
gcc -c progn.c -o progn.o
```

2.3.1 Création une librairie statique

Pour créer une librairie statique à partir des fichiers objet, il faut utiliser la commande `ar` qui archive ces fichiers dans un seul fichier. L'option -r permet d'insérer les nouveaux fichiers dans l'archive. L'option -v (verbose) permet d'afficher à l'écran le nom des fichiers insérés.

```
ar -rv libtoto.a prog1.o prog2.o prog3.o ... progn.o
```

La librairie `libtoto.a` est prête à être utilisée dans une compilation.

[1] Sous linux, cette variable n'est pas prise en compte, il faut que le nom de la librairie se trouve dans le fichier `/etc/ld.so.conf` et que `ldconfig` ait été lancé après la modification de ce fichier.

2.3.2 Création une librairie dynamique

Pour créer une librairie statique à partir des fichiers objet, on peut utiliser `gcc` avec l'option `-shared`.

```
gcc -o libtoto.so -shared prog1.o prog2.o prog3.o ... progn.o
```

La librairie `libtoto.so` est prête à être utilisée dans une compilation.

2.3.3 L'emplacement d'une librairie

En général, on place une librairie (ou un lien vers cette librairie) à un emplacement visible par tous les programmes qui sont susceptibles de l'utiliser. Typiquement dans :

- `/usr/local/lib` si la librairie est susceptible d'être utilisée par plusieurs utilisateurs ;
- `~/lib` si la librairie est susceptible d'être utilisée par un seul utilisateur.

Dans le cas d'une librairie dynamique, il faudra penser à vérifier que la variable d'environnement `LD_LIBRARY_PATH` permettra aux programmes de trouver la nouvelle librairie.