bibliothèque BFD

23 septembre 2009

1 Sans utiliser de bibliothèque

La commande UNIX *nm* consulte ce qu'on appelle la *table des symboles* d'un fichier objet, d'une bibliothèque ou d'un exécutable. Voici un exemple.

```
$ cat a.c
int f ()
{
    return 1;
}

int g ()
{
    return 2;
}

$ cat b.c
float zero = (float)0;

char un ()
{
    return '1';
}

$ cat main.c
extern int f ();
extern int g ();
int main ()
{   int x;
    x = f ();
}

$ gcc -c a.c b.c main.c
$ nm -g a.o
0000000000000000 T f
0000000000000014 T g
$ nm -g b.o
0000000000000014 T un
0000000000000000 D zero
$ nm -g main.o
                 U f
0000000000000000 T main
$ gcc a.o b.o main.o
$ nm -g a.out
... bla bla bla ...
0000000100000764 T f
0000000100000778 T g
00000001000007b4 T main
00000001000007a0 T un
000000010000078c D zero
```

Avec l'option *-g nm* n'imprime que les symboles *globaux* (ou *extern* en C). On s'intéresse aux deux dernières colonnes. On voit que

1. *a.o* comporte deux symboles globaux f et g. La lettre T indique que les symboles sont définis dans des sections *text* du fichier objet (les sections *text* contiennent le code en langage machine ; on a donc affaire à des fonctions) ;
2. *b.o* comporte deux symboles *zero* et *un*. La lettre D indique que les symbole *zero* est défini dans une section *data* du fichier objet. Les sections *data* contiennent les données initialisées. On a donc affaire à une variable initialisée, à une chaîne de caractères etc ...
3. *main.o* comporte un symbole *main* et une référence indéfinie (la lettre U signifie undefined) à f.

L'édition des liens est possible ici. L'exécutable est obtenu en *juxtaposant* le contenu des trois fichiers objets. On remarque que le contenu du fichier *b.o* est présent bien qu'il ne soit pas en fait utilisé. L'exécutable ne comporte aucune référence indéfinie. Résumons–nous.

Proposition 1 *Dans le cas où on n'utilise aucune bibliothèque, l'édition des liens entre plusieurs fichiers objets est possible si, et seulement si,*

1. *aucun symbole global défini dans un fichier objet n'est redéfini dans un autre ;*
2. *le symbole* main *est défini ;*
3. *chaque symbole global indéfini dans un fichier objet est défini dans un autre.*

2 Utilisation d'une bibliothèque

Une bibliothèque (fichier *.a* ou *archive file* en Anglais) n'est rien d'autre qu'une juxtaposition de fichiers objets (plus un index pour s'y retrouver plus facilement). On les crée avec la commande *ar* (pour *archive*). Suite de l'exemple précédent.

```
$ ar cr libapin.a a.o b.o
$ gcc -L. main.c -lapin
```

L'appel à *ar* a créé la bibliothèque *libapin.a*. Cette bibliothèque contient les deux fichiers objets *a.o* et *b.o*. Sur la dernière ligne, on a compilé *main.c* en éditant les liens avec cette bibliothèque. Quelques remarques sur les options de *gcc* :

1. l'option *-Ldirectory* indique à *gcc* qu'il faut chercher la bibliothèque dans le répertoire *directory* (sur l'exemple, il faut chercher dans le répertoire courant .) ;
2. pour éditer les liens avec *libxxx.a* on passe à *gcc* l'option *-lxxx* (d'où l'écriture *-lapin*).

La commande *nm* permet aussi d'étudier le contenu d'une bibliothèque. Suite de l'exemple.

```
$ nm -gs libapin.a

Archive index:
f in a.o
g in a.o
zero in b.o
un in b.o

a.o:
0000000000000000 T f
0000000000000014 T g

b.o:
0000000000000014 T un
0000000000000000 D zero
$ nm -g a.out
... meme bla bla ...
0000000100000784 T f
0000000100000798 T g
0000000100000764 T main
```

L'option *-s* passée à *nm* n'est utile que dans le cas d'une bibliothèque : elle demande l'impression de la table d'index. On remarque que le contenu du fichier *b.o* n'est pas inclus dans l'exécutable. C'est dû au fait qu'aucun symbole défini dans *b.o* n'est nécessaire.

Proposition 2 *Lorsqu'on édite les liens avec une bibliothèque, tout se passe comme si on éditait les liens avec les fichiers objets qui la composent et dont au moins un symbole est nécessaire.*

3 La bibliothèque standard

Sauf demande expresse du contraire, *gcc* provoque systématiquement l'édition des liens avec la bibliothèque standard *libc.a* habituellement située dans le répertoire /usr/lib.

```
$ nm -g /usr/lib/libc.a
... gros bla bla ...
printf.o:
         U ___errno
         U __iob
         U __threaded
         U _doprnt
         U _ferror_unlocked
         U _flockget
         U _mutex_unlock
         U _setorientation
00000000 T printf
... gros bla bla ...
```

4 Utilisation de plusieurs bibliothèques

On peut fort bien être amené à utiliser plusieurs bibliothèques et il se peut que certains fichiers objets contenus dans une bibliothèque aient des références indéfinies vers des symboles définis dans une autre bibliothèque.

Une variante de l'exemple précédent illustre cette situation.

```
$ cat x.c
extern int trois;
int quatre ()
{
    return trois + 1;
}
$ cat y.c
int trois = 3;
$ cat princ.c
#include <stdio.h>
extern int quatre ();
int main ()
{
    printf ("%d\n", quatre ());
}
$ gcc -c x.c y.c princ.c
$ ar cr liboup.a x.o
$ ar cr libubrique.a y.o
$ nm -gs liboup.a
Archive index:
quatre in x.o

x.o:
0000000000000000 T quatre
                 U trois
$ nm -gs libubrique.a
```

```
Archive index:
trois in y.o

y.o:
0000000000000000 D trois
$ nm -g princ.o
0000000000000000 T main
                 U printf
                 U quatre
$ gcc -L. princ.o -loup -lubrique
$ a.out
4
$ gcc -L. princ.o -lubrique -loup
./liboup.a(x.o): In function ‘quatre’:
x.o(.text+0x4): undefined reference to ‘trois’
x.o(.text+0x8): undefined reference to ‘trois’
x.o(.text+0xc): undefined reference to ‘trois’
x.o(.text+0x18): undefined reference to ‘trois’
collect2: ld returned 1 exit status
```

On remarque que l'ordre dans lequel les bibliothèques sont passées à *gcc* est important : l'éditeur de liens parcourt les bibliothèques dans l'ordre dans lequel elles sont données. Il ne revient pas *en arrière*.

Dans le premier cas, l'édition des liens se passe bien. Lors du parcours de *liboup.a*, le fichier *x.o* est inclus dans l'exécutable parce qu'il contient la définition manquante de *quatre*. Ensuite, lors du parcours de *libubrique.a*, le fichier *y.o* est inclus parce qu'il contient la définition manquante de *trois*.

Dans le deuxième cas, l'édition des liens échoue. Lors du parcours de *libubrique.a* l'éditeur des liens n'inclut pas le fichier *y.o* parce qu'il n'y a encore aucune référence indéfinie vers *trois*. Ensuite, lors du parcours de *liboup.a*, le fichier *x.o* est inclus. C'est seulement à ce moment–là qu'une référence indéfinie vers le symbole *trois* apparaît. Comme la bibliothèque *libubrique.a* n'est pas parcourue une deuxième fois, la référence reste indéfinie et l'édition des liens échoue.

5 Le projet

6 Le programme demandé

Votre programme *projet* doit respecter la syntaxe suivante.

```
projet [--objets|--symboles] fichier-1 ... fichier-n
```

Il retourne le code 0 si l'édition des liens est possible et un code différent de 0 sinon. Votre programme doit (comme *gcc*) systématiquement faire l'édition des liens avec la bibliothèque standard. Si l'édition des liens est impossible, il doit indiquer par un message approprié sur la sortie d'erreur standard les raisons de l'échec. En cas de succès, si le premier paramètre vaut *–objets* il doit imprimer les noms des fichiers objets nécessaires à l'édition des liens (y compris ceux qui sont inclus dans les bibliothèques) ; si le premier paramètre vaut *–symboles* il doit imprimer les symboles globaux qui figureront dans l'exécutable.

On a pu réaliser le projet en répartissant le code sur deux fichiers *projet.c* et *symtab.c* qui totalisent un peu plus de 300 lignes. Voici des exemples d'exécution.

```
$ projet main.o a.o b.o
$ projet --objets main.o a.o b.o
main.o
a.o
b.o
$ projet --symboles main.o b.o a.o
main
f
zero
un
```

```
g
$ projet a.o b.o
main is undefined
linking failed
$ projet a.o a.o
twice defined symbol: f
twice defined symbol: g
main is undefined
linking failed
$ projet --objets main.o libapin.a
main.o
a.o
$ projet princ.o liboup.a libubrique.a
$ projet princ.o libubrique.a liboup.a
undefined symbol: trois
linking failed
$ projet projet.o symtab.o /usr/lib/libbfd.a /usr/lib/libiberty.a
$ projet projet.o symtab.o /usr/lib/libiberty.a /usr/lib/libbfd.a
undefined symbol: xstrerror
undefined symbol: xexit
undefined symbol: objalloc_create
undefined symbol: objalloc_free
undefined symbol: _objalloc_alloc
undefined symbol: objalloc_free_block
undefined symbol: hex_init
undefined symbol: _hex_value
undefined symbol: concat
linking failed
```

7 La documentation demandée

Avec le programme C on vous demande une documentation qui ne doit pas dépasser 10 pages. Inutile d'inclure un listing du programme. Cette documentation doit décrire

1. les algorithmes que vous mettez en œuvre pour résoudre le problème posé (gestion de la table des symboles ...) ;
2. une documentation aussi précise que possible des fonctions de la bfd que vous utilisez (cf. ci–dessous). Il est donc déconseillé d'utiliser plus de fonctions de la bfd qu'il n'est strictement nécessaire (éviter les gros *copier–coller*).

8 Informations utiles

Vous utiliserez la bibliothèque *libfd.a*. Vous aurez probablement besoin d'inclure le fichier *bfd.h* et d'éditer les liens avec (dans l'ordre, cf. ci–dessus) les bibliothèques *libfd.a* et *libiberty.a*.

Pour trouver la bonne façon d'utiliser la bibliothèque, le plus simple consiste à étudier le contenu du fichier *bfd.h* et le code du programme *nm.c*. On conseille de tracer l'exécution de *nm* en utilisant le debugger.

Dans /home/enseign/boulier/binutils-2.9/binutils se trouvent le fichier *nm.c* ainsi que l'exécutable *nm* compilé de façon à pouvoir en tracer l'exécution au debugger. On y trouve aussi une documentation *bfd.dvi*. On vous déconseille de l'imprimer (gros fichier). Méfiez–vous en : elle ne correspond pas exactement à la version de la bfd que vous utilisez ; en plus elle contient apparemment des erreurs.

8.1 Contourner certaines erreurs de la bfd

Pour contourner certaines erreurs (semble–t–il) de la bibliothèque bfd (qui surviennent lorsqu'on parcourt les bibliothèques prédéfinies), on conseille d'utiliser les tests suivants pour déterminer si un symbole est global ou indéfini :

```
bfd* fichier_objet;   /* fichier objet courant */
asymbol* sym;         /* symbole courant dans le fichier_objet */
symbol_info syminfo;

...
bfd_get_symbol_info (fichier_objet, sym, &syminfo);
if (isupper ((unsigned char)syminfo.type)
{
    /* le symbole est global */
}
if (syminfo.type == 'U')
{
    /* le symbole est {\`a} la fois global et ind{\'e}fini */
}
```

9 Progression conseillée

On vous conseille de réaliser votre projet un peu à la fois en incluant de plus en plus de fonctionnalités. Voici une progression proposée :

1. Écrire un programme qui lit un nom de fichier objet sur sa ligne de commande et qui imprime la table de ses symboles (comme *nm*).
2. Même chose mais en traitant en plus le cas d'une bibliothèque.
3. Réaliser le projet dans le cas où aucune bibliothèque n'est utilisée.
4. Réaliser le projet dans le cas où seule la bibliothèque standard est utilisée.
5. Réaliser le projet complet.

Attention : testez votre programme sur les machines du M5.

10 Évaluation

Le programme compte pour 1/3 de la note. Le rapport pour 1/3. Un contrôle TP sur machine pour 1/3.

Il est indispensable que votre programme soit compilable et respecte la syntaxe donnée dans l'énoncé.

Un programme qui résout parfaitement les problèmes simples et pas du tout les problèmes compliqués est préféré à un programme qui résout tout à moitié.

La note tiendra compte de la lisibilité du code et de la lisibilité des identificateurs de variables et de fonctions.