Édition de lien, fichiers ELF

Ensimag 1A Apprentissage, Mai 2010

Exemple

Un programme, deux fichiers :

ppal.s

```
        .text
        .global main
main:
        push %ebp
        movl %esp, %ebp

        call dire_bonjour

        leave
        ret
```

bonjour.s

```
        .data
bonjour: .string "bonjour\n"
        .text
        .global dire_bonjour
dire_bonjour:
        push %ebp
        movl %esp, %ebp

        pushl $bonjour
        call printf
        addl $4, %esp

        leave
        ret
```


Génération d'un exécutable

Assemblage :

```
gcc -c ppal.s -o ppal.o # crée ppal.o
gcc -c bonjour.s -o bonjour.o # crée bonjour.o
```

À ce stade, **`ppal.o`** ne connait pas encore l'adresse de la fonction “dire_bonjour”

Édition de liens :

```
gcc ppal.o bonjour.o -o mon_programme
```


Contenu des fichiers *.o

La commande “nm” permet de connaitre les symboles utilisés et définis par un binaire :

```
$ nm ppal.o bonjour.o
bonjour.o:
00000000 d bonjour            "d" comme "data"
00000000 T dire_bonjour       "T" comme "text"
         U printf             "U" comme "undefined"
ppal.o:
         U dire_bonjour
00000000 T main
```


Définition

Liaison = toute opération qui établit tout ou partie de la chaîne d'accès qui permet de passer du nom d'un objet informatique à sa représentation physique

Caractéristiques de la liaison (1)

Liaison d'un objet lors de la traduction du programme

Liaison partielle

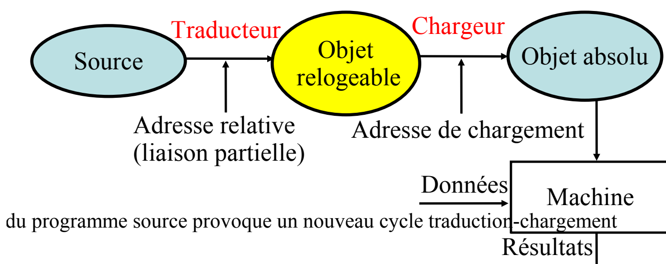
Compilation : les identificateurs sont remplacés par des adresses relatives à l'origine de blocs bien identifiés (section data, text par exemple)

Ex. : bonjour remplacé par @.data+0x00

Le programme produit n'est plus directement exécutable et doit être traité lors d'une phase d'édition de liens/chargement

- Edition de liens : cette phase a pour but d'établir la liaison des références externes (références à des objets de bibliothèques de programmes ou à des objets définis dans des modules compilés séparément)

Etape de la vie d'un programme programme unique



Etape de la vie d'un programme programme composé

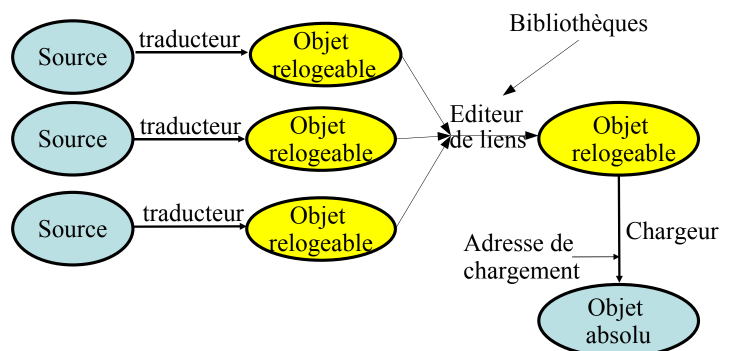


Table de relocation, table des symboles

Table des symboles

Liste des adresses (relatives) des symboles dans leur section

Rq : si plusieurs sections de même type, elles sont fusionnées au préalable

Table de relocation/relogement

Liste des trous dans le code à remplir (en utilisant la table de symboles)

Structures de données

Table des symboles

Section	@relative	nom symbole
	undef	

Table de relogement (adresse des modifications à faire dans le code)

Adresse du trou	Section symbole	Nom symbole

Exemple

Code :

```
1                       .section .data
2 0000 03000000 i:      .int 3
3 0004 FF       j:      .byte 0xff
4                       .section .text
5                       .global main
6 0000 B8000000 main:   movl $i,%eax
6      00
7 0005 3A050400         cmpb j,%al
7      0000
8 000b 3D000000         cmpl $main,%eax
8      00
9 0010 C3               ret
```

Table des symboles :

data	0x00	i
data	0x04	j
text	0x00	main

Table de reloc :

0x01	text	.data (-> i)
0x07	text	.data (-> j)
0x0c	text	main

Exemple (2)

Table de relogement :

AD		S
0x01	text	.data (-> i)
0x07	text	.data (-> j)
0x0c	text	main

Chaque ligne = 1 “trou” dans le fichier relogeable

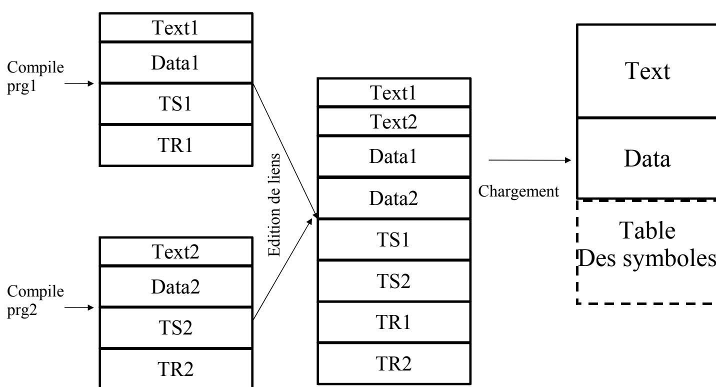
Relogement :

emplacement(AD) := emplacement(AD) + adresse_finale_de(S)

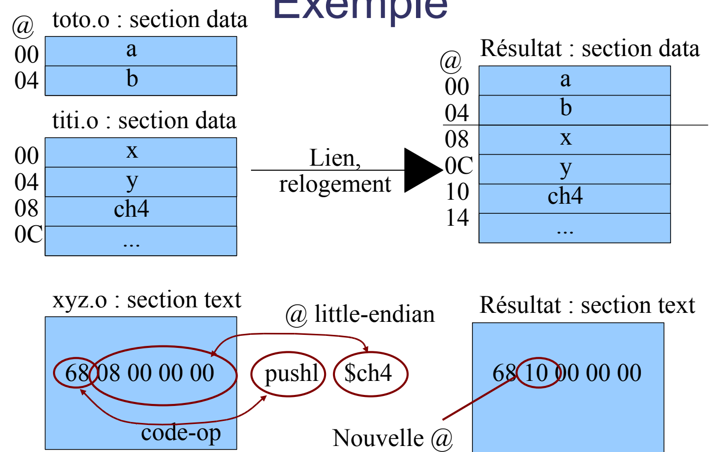
En fait, plusieurs types de relogements

Cf. documentation du projet.

Structure de donnée (prg composé)



Exemple



Fonctions d’un chargeur

Transformer «l’objet relogeable» en «objet absolu»

```
     .section data
adtab: .long  tab      # tab est relative à la section bss
adsom: .long  som      # som est relative à la section text
x:     .long  3
     .section bss
…
       .lcomm tab, 10
…
     .section text
…
som:   enter  $10,0
       pushl  $x       # x est relatif à la section data
…
```

tab, som, x sont dites des adresses translatables

Si c’est un «load and go» on doit déterminer l’adresse absolue qui doit recevoir le contrôle

Le format ELF : motivations

Types de fichiers utilisés dans la chaine de compilation :

- Source de haut niveau (C, ...)
- Assembleur (fichier.s)
- Objet (fichier.o)
- Bibliothèques partagées (fichier.so)
- Executables

Besoins similaires pour .o, .so et executables :

- Code compilé
- Différentes sections (.data, .text, ...)
- Tables (de symbole, de relogement)

Le format ELF : principe

ELF : Executable and Linkable Format

Format commun aux .o, .so et exécutables

Principe de ELF :

- Stoquer dans le même fichier toutes les informations (sections, tables de symboles, ...)
- Séquentialisation (1 fichier = 1 suite d'octets)

Concretement :

- Un en-tête qui donne les adresses des sections suivantes
- Les sections, les unes après les autres

Le format ELF : Informations supplémentaires

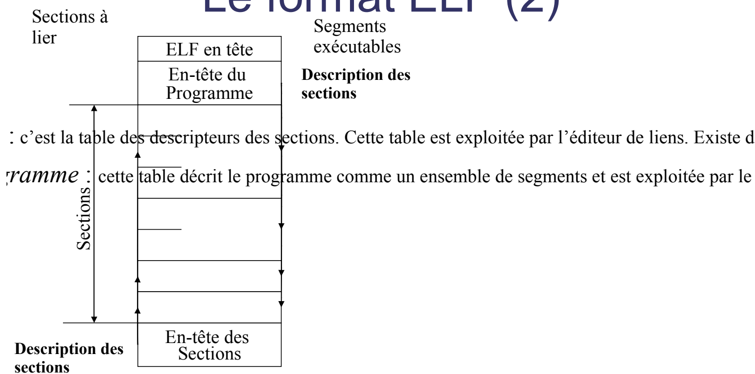
Motivation :

- permettre la liaison dynamique et notamment faciliter la gestion à l'exécution du langage C++
- Remplacement du Common Object File Format (COFF, qui était le format d'Unix système V)
- Format actuel d'Unix système V, Linux, et de nombreuses variantes d'Unix BSD (mais pas Mac OS X qui utilise Mach-O)

Un ficher ELF peut être relogeable, exécutable ou partageable

- Relogeable => doit être traité par l'éditeur de liens
- Exécutable => a été relogé, et a tous ses symboles résolus sauf peut être les références aux bibliothèques partagées qui sont résolues à l'exécution
- Partageable => une bibliothèque partagée

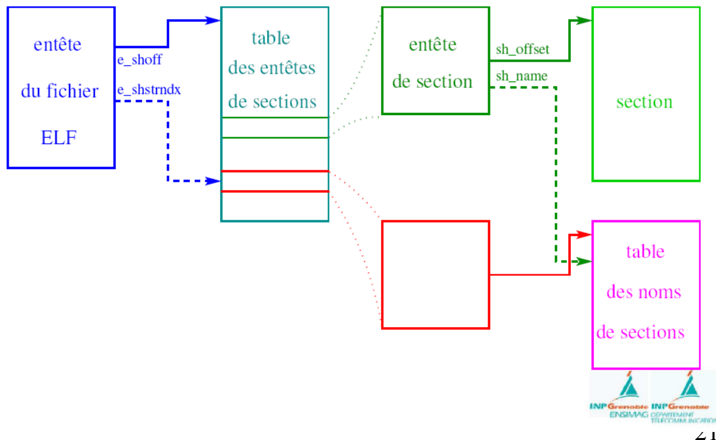
Le format ELF (2)



Le format ELF (3)

- L'entête d'un fichier ELF (type C : `struct Elf32_Ehdr`) donne des informations sur le format et le nombre de sections du fichier
- Il possède aussi un "pointeur" vers la table des entêtes de sections : `e_shoff`
  - ⇝ *"pointeur" = valeur indiquant le déplacement par rapport au début du fichier*
- Entête de section (type C : `struct Elf32_Shdr`) : infos sur la taille de la section, son emplacement dans le fichier, son type, . . .
- Types de sections : `.text`, `.data`, `.bss`, table des symboles, tables des chaînes (noms de sections, noms de symboles), tables de relocations, . . .

Le format ELF (4)



Le format ELF : table des chaines, tables des symboles

Table des chaines : chaines (noms de symboles, ...) concaténées (séparateur : '\0')

'\0'	v	a	r	'\0'	m	a	i	n	'\0'	f	i	n	i	f	'\0'
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Table des symboles : liste des symboles définis/indéfinis :

- Nom (indice dans table des chaines)
- Numéro (utilisé dans la table de relocation)
- Section (indice dans la table des sections)
- Adresse relative au début de section
- ...

Format ELF : l'en-tête du fichier

```
char magic[4];      // tableau initialisé avec \177ELF
char class;         // taille d'une adresse 1 pour 32 bit et 2  pour 64 bit
char byteorder;     // 1 little endian, 2 big endian
char hversion;      // version de l'en-tête toujour 1.
char pad[9];
short filetype; // 1 relogeable, 2 exécutable, 3 partageable, 4 image mém.
short archtype; // 2 Sparc, 3 x86, 4 68K, …
int fversion;   // toujours 1
int entry;      // point d'entrée si c'est un exécutable
int phdrpos, shdrpos; // position dans le fichier des en-têtes programme et segment ou 0
int flags;          // fanion propre à certaine architecture, en général 0
short hdrsize;      // taille de cet en-tête
short phdrent;// taille d'une entrée de la table des en-têtes programme
short phdrcnt;// nombre d'entrées dans la table précédente ou 0
short shdrent;// taille d'une entrée dans la table des en-tête de section
short shdrcnt;// nombre d'entrées dans la table précédente ou 0
short strsec; // numéro de la section qui contient les noms de sections.
```


Format ELF : en-tête de section

```
int sh_name;   /* index dans la table des chaînes */
int sh_type;   /* type de section */
int sh_flags;  /* 3 bits utilisés :
                  ALLOC, WRITE, EXEINST */
int sh_addr;   /* adresse de base en mémoire
                  si chargeable ou 0 */
int sh_offset;    /* déplacement dans le fichier
                     du début de la section */
int sh_size;   /* taille en octets */
int sh_info;   /* information spécifique à la section */
int sh_align;  /* granularité de l'alignement
                  si la section est déplacée */
int sh_entsize/* taille d'une entrée
                  si la section est un tableau */
```

Format ELF : types des sections 1

Le champ sh_type de section inclut les types suivant :

PROGBIT: la section peut contenir du code des données et des information de mise au point.

NOBIT : identique à PROGBIT mais aucune mémoire n'est allouée dans le fichier. Est utilisé pour la section bss.

SYMTAB et DYNSYM : la section contient une table des symboles soit pour la liaison statique soit pour la liaison dynamique.

STRTAB : la liaison contient les noms des symboles qui sont en général spécialisé (nom de section, symbole pour l'édition de lien dynamique, …).

REL et RELA : la section contient des informations de relogement. REL provoque l'addition de la valeur de relogement à la valeur de base stockée dans le code ou les données, RELA inclut la valeur de relogement dans l'entrée

DYNAMIC and HASH contient des informations pour l'éditeur de liens dynamique

Format ELF : types des sections 2

Un fichier relogeable exécutable contient une douzaine de section :

.text de type PROGBIT avec les attribut ALLOC+EXECINSTR

.data de type PROGBIT avec les attribut ALLOC+WRITE

.rodata de type PROGBIT avec l'attribut ALLOC

.bss de type NOBIT avec les attribut ALLOC+WRITE

.rel.text, .rel.data, et .rel.rodata chacune est de type REL ou RELA et contient les informations permettant de reloger le code ou les données.

.init et .fini chacune de type PROGBIT avec les attribut ALLOC+EXECINSTR

.symtab et .dynsym respectivement de type SYMTAB et DYNSYM. La section .dynsym a l'attribut ALLOC positionné.

.strtab et .dynstr toutes les deux de type STRTAB. .dynstr a l'attribut ALLOC positionné.

.got et .plt ces deux sections sont utilisés pour la liaison dynamique

.debug, .line (association ligne source, code), .comment

Format ELF : table des symboles

```
int  name;     /* position du nom dans la table
                  des chaînes */
int value;     /* valeur du symbole, relative à la
                  section dans le fichier relogeable */
int size;      /* taille de l'objet ou de la fonction */
char type;     /* donnée, fonction, section, ou
                  spécial (nom du fichier source) */
char bind;     /* symbole local, global ou faible */
char other;       /* pas utilisé */
short sect;       /* numéro de la section,
                  ABS, COMMON, UNDEF */
```


Format ELF : en-tête programme

```
int  type;     /* code, ou données chargeables, info pour
                  l'édition de liens dyna. */
int offset;    /* déplacement dans le fichier du segment */
int virtaddr;  /* adresse de chargement du segment */
int physaddr;  /* pas utilisé */
int filesize;  /* taille du segment dans le fichier */
int memsize;   /* taille du segment en mémoire (bss…) */
int flags;     /* Read, Write, Execute */
int align      /* alignement requis */
```