Environnement de Programmation et Programmation Système

Gilles Menez

Université de Nice – Sophia-Antipolis
Département d'Informatique
email : menez@unice.fr
www : www.i3s.unice.fr/~menez

18 septembre 2009: V 1.1

Table des Matières (1)

Table des Matières (2)

Références

J.P. Braquelaire
Méthodologie de la programmation en Langage C - Principes et Applications.
Masson, 1995, 2e édition.

Kernighan B.W., Ritchie D.M
Le Langage C - C ANSI.
Masson, Prentice Hall, 1994, 2e édition.

P.Collet
Bibliothèques
2007-2008 Université de Nice Sophia Antipolis

P.Collet
Construction de programmes - Make et autres outils
2007-2008 Université de Nice Sophia Antipolis

Bibliothèques

Ecrire avec les autres . . .

Un logiciel est un agglomérat de morceaux de code écrits par différents organismes eux mêmes constitués de différentes personnes et qui, une fois assemblés correctement vont collaborer pour produire les actions escomptées.

- ➟ Ceci est particulièrement vrai dans les applications ”système” qui utilisent des concepts présents et **écrits par d’autres** dans le système d’exploitation.

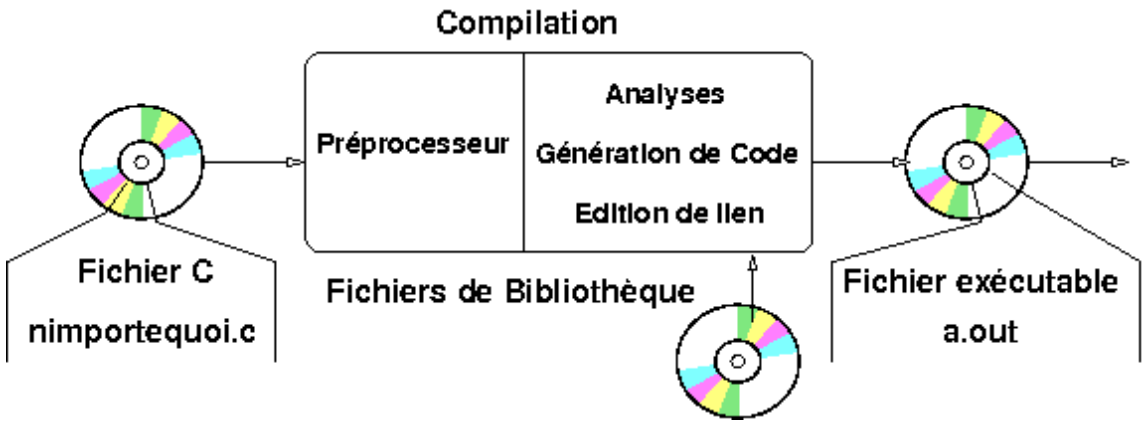
Une **bibliothèque** permet de regrouper un ensemble de fonctions de base utilisables et utilisées par de nombreux programmes.

- ➟ C’est souvent ”l’unité d’échange” entre les personnes :

  ”J’ai développé ce concept, j’en ai fait une bibliothèque !”

Bibliothèques : Une démarche de « génie logiciel »

- **Réutilisation** : Un ”bon” programmeur est ”fénéant” :-)
- **Fiabilité** : Puisque la bibliothèque a été testée de nombreuses fois et que l'on en bénéficie.
- **Portabilité** : Le souhait de partager nécessite d'ecrire du code portable et compréhensible. Une bibliothèque connue remplit ces critères.
- **Efficacité** : La bibliothèque contient une forme ”compilée” du programme ce qui permet d'optimiser la compilation globale.



Bibliothèques en C

Le langage C fait le choix de placer en bibliothèque

① tout ce qui ne concerne pas le langage :
- ➢ les entrées-sorties,
- ➢ la gestion mémoire,
- ➢ les chaînes de caractères, . . .

② et qui peut être considéré comme une unité de réutilisation : des fonctions, des types, des constantes.

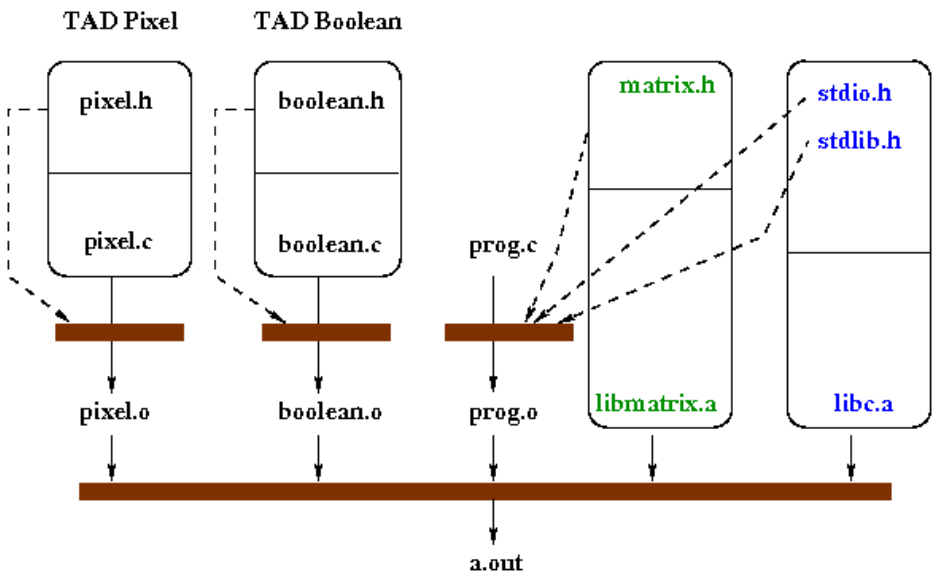
Ce choix n'est pas un hasard [**?**] :

- ➢ Cela permet d'alléger le compilateur (constructions syntaxiques et sémantiques sont en nombre réduit).
- ➢ C'est aussi un moyen efficace pour gérer les extensions de fonctionnalités.

Unités de compilation et Interfaces

Rappel :

Le schéma de compilation est arborescent faisant apparaître des transitions et des dépendances de différentes natures.



- Les feuilles sont les unités de compilation : ".c"
- Les ".h" sont les **interfaces** des unités de compilation.

La compilation de chaque branche peut être menée de façon séparée, d'où la notion de **compilation séparée** !

Archive de fichiers

La notion de bibliothèque informatique est plus précise qu'un "lieu où on viendrait chercher une information".

➠ Sinon, à ce titre un fichier `objet` serait une bibliothèque . . .

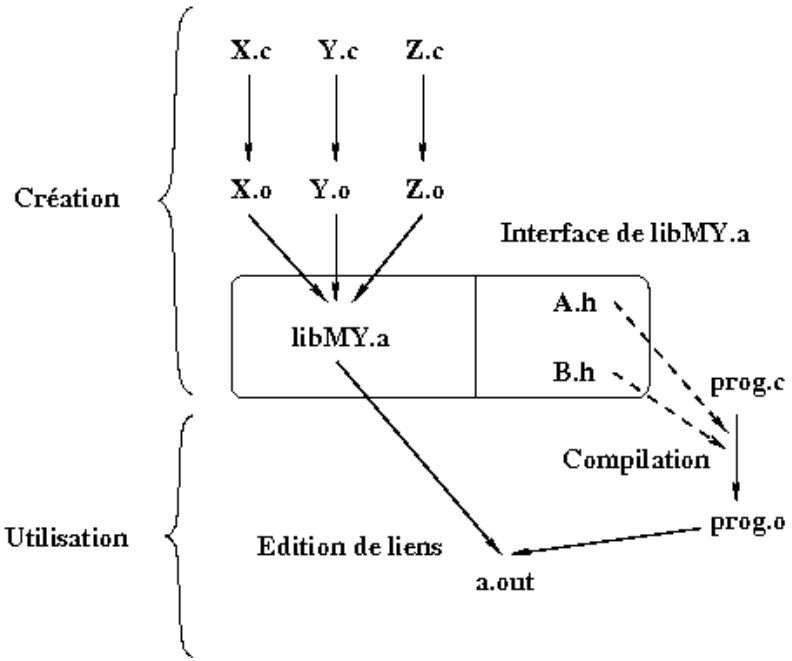
Une bibliothèque est une **archive** de fichiers objets.

➠ En tant qu'archive, elle induit une organisation précise de façon à permettre la récupération des fichiers originaux appelés **membres de l'archive**.

Toutefois, lors de l'invocation du compilateur, "ce n'est qu'un ensemble organisé de fichiers objets" qui seront liés en même temps que les autres par **l'édition de liens** .

Cycle de vie d'une Bibliothèque

La prise en compte d'une bibliothèque par le schéma de compilation nécessite la création d'un fichier d'archive **ET** de son ou de ses fichiers d'interfaces (i.e header files ou fichiers d'entête).



- Dans le code source, chaque occurence d'un appel de fonction de bibliothèque nécéssitant à la compilation une vérification sémantique, il est indispensable qu'il soit précédé par la déclaration de la fonction.

  C'est le rôle des fichiers d'entête qui permettent (du fait de leur inclusion) à la compilation de mettre en correspondance utilisation (i.e appel) et déclaration (i.e prototype).

- Le fichier d'entête ne contient pas (classiquement) d'information nécessitant la génération de code.

  La mise en correspondance de l'appel d'une fonction avec le corps appelé se fait à l'édition de lien et nécessite la fourniture de bibliothèque adéquate.

Le C : Un effort de standardisation

"The C programming language, before it was standardized, did not provide built-in functionalities such as I/O operations (unlike traditional languages such as Cobol and Fortran) . . .

Many universities and organizations began creating their own variations of the language for their own projects. By the beginning of the 1980s compatibility problems between the various C implementations became apparent."

En 1983, l'**American National Standards Institute (ANSI)** forme un comité pour établir une spécification standard de C connue sous **ANSI C** .

Ce travail aboutit en 1989 à la création du standard C89 :

- Ce standard comprend un ensemble de bibliothèques logicielles :

```
"the ANSI C standard library".
```

Bibliothèque ”ISO C”

Des révisions ultérieures ajoutent des notions et donc des header files.

- Leurs supports varient entre les implémentations des compilateurs.

The headers <iso646.h>, <wchar.h>, and <wctype.h> were added with ***Normative Addendum 1*** *(NA1), an addition to the C Standard ratified in 1995.*

The headers <complex.h>, <fenv.h>, <inttypes.h>, <stdbool.h>, <stdint.h>, and <tgmath.h> were added with ***C99****, a revision to the C Standard published in 1999.*

The ISO C standard library consists of 24 C header files which can be included into a programmer’s project with a single directive.

- Each header file contains one or more function declarations, data type definitions and macros.

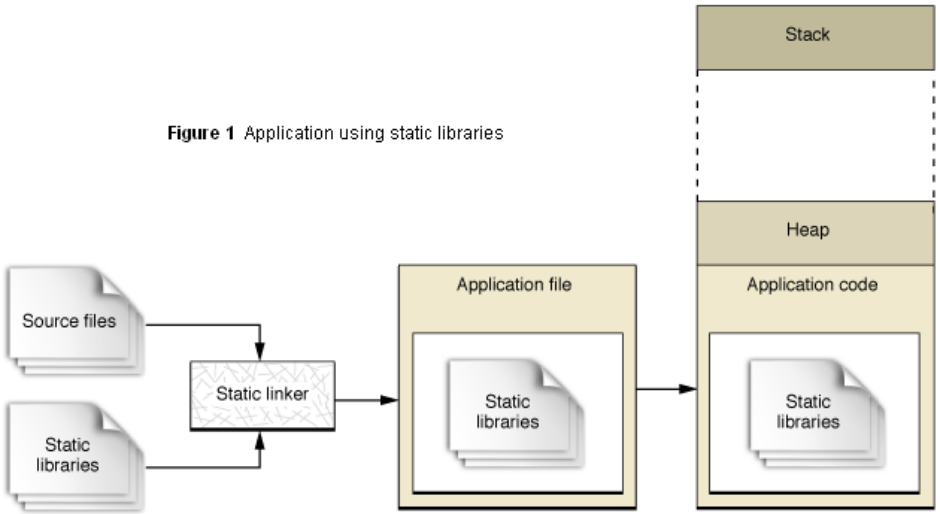
ISO C library headers

Fichiers d'en-tête	Thématiques : (http://en.wikipedia.org/wiki/C_standard_library)
<assert.h>	Contains the assert macro, used to assist with detecting logical errors and other types of bug in debugging versions of a program.
<complex.h>	A set of functions for manipulating complex numbers. (New with C99)
<ctype.h>	Contains functions used to classify characters by their types or to convert between upper and lower case in a way that is independent of the used character set (typically ASCII or one of its extensions, although implementations utilizing EBCDIC are also known).
<errno.h>	For testing error codes reported by library functions.
<fenv.h>	For controlling floating-point environment. (New with C99)
<float.h>	Contains defined constants specifying the implementation-specific properties of the floating-point library, such as the minimum difference between two different floating-point numbers (_EPSILON), the maximum number of digits of accuracy (_DIG) and the range of numbers which can be represented (_MIN, _MAX).
<inttypes.h>	For precise conversion between integer types. (New with C99)
<iso646.h>	For programming in ISO 646 variant character sets. (New with NA1)
<limits.h>	Contains defined constants specifying the implementation-specific properties of the integer types, such as the range of numbers which can be represented (_MIN, _MAX).
<locale.h>	For setlocale() and related constants. This is used to choose an appropriate locale.
<math.h>	For computing common mathematical functions
<setjmp.h>	Declares the macros setjmp and longjmp, which are used for non-local exits
<signal.h>	For controlling various exceptional conditions
<stdarg.h>	For accessing a varying number of arguments passed to functions.
<stdbool.h>	For a boolean data type. (New with C99)
<stdint.h>	For defining various integer types. (New with C99)
<stddef.h>	For defining several useful types and macros.
<stdio.h>	Provides the core input and output capabilities of the C language. This file includes the venerable printf function.
<stdlib.h>	For performing a variety of operations, including conversion, pseudo-random numbers, memory allocation, process control, environment, signalling, searching, and sorting.
<string.h>	For manipulating several kinds of strings.
<tgmath.h>	For type-generic mathematical functions. (New with C99)
<time.h>	For converting between various time and date formats.
<wchar.h>	For manipulating wide streams and several kinds of strings using wide characters - key to supporting a range of languages. (New with NA1)
<wctype.h>	For classifying wide characters. (New with NA1)

Caractérisation

Une bibliothèque **statique** est une archive dont les membres sont des `fichiers objets` (.o)

➟ Elle contient donc du code exécutable !



Le fichier exécutable lié avec une telle bibliothèque contient le code de la bibliothèque et souvent dans son intégralité !

➟ Naturellement, à l'exécution, l'espace d'adressage du processus contient aussi le code !

Avantages/Inconvénients

No Good	Good
Taille de l'exécutable	L'exécutable est "self containing" !
Temps de chargement	Le plus simple pour l'OS !
Si la bibliothèque évolue, il faut recompiler !	Et aussi peut être le plus rapide ? (édition de lien au "compile-time" au lieu du "run-time")

Pourquoi et Comment "Statique" ?

Pour comprendre, il faut analyser le code généré (`gcc -static`) :

```
1   #include <math.h>
2   int main(int argc, char * argv[]){
3     float x,y;
4     y = sin(x);
5   }
```

```
1   as.out:     file format elf32-i386
2
3   Disassembly of section .text:
4
5   08048228 <main>:
6     ...
7     804823c:      fstpl   (%esp)
8     804823f:      call    8048250 <_sin>
9     8048244:      fstps   0xfffffff8(%ebp)
10    ...
11    804824b:      pop     %ebp
12    804824c:      lea     0xfffffffc(%ecx),%esp
13    804824f:      ret
14
15  08048250 <_sin>:
16    ...
17    8048271:      fstp    %st(1)
18    8048273:      fsin
19    8048275:      ret
20    8048276:      nop
```

- Chaque fonction se situe à un offset statique du début de fichier.
- L'appel à une fonction fait référence à cet offset.
- Toutes les fonctions de la libc sont dans le fichier (d'où la taille !).

Manipulations : Commande ar

Le **nommage** d'une bibliothèque statique devrait être de la forme :

lib*nom*.a

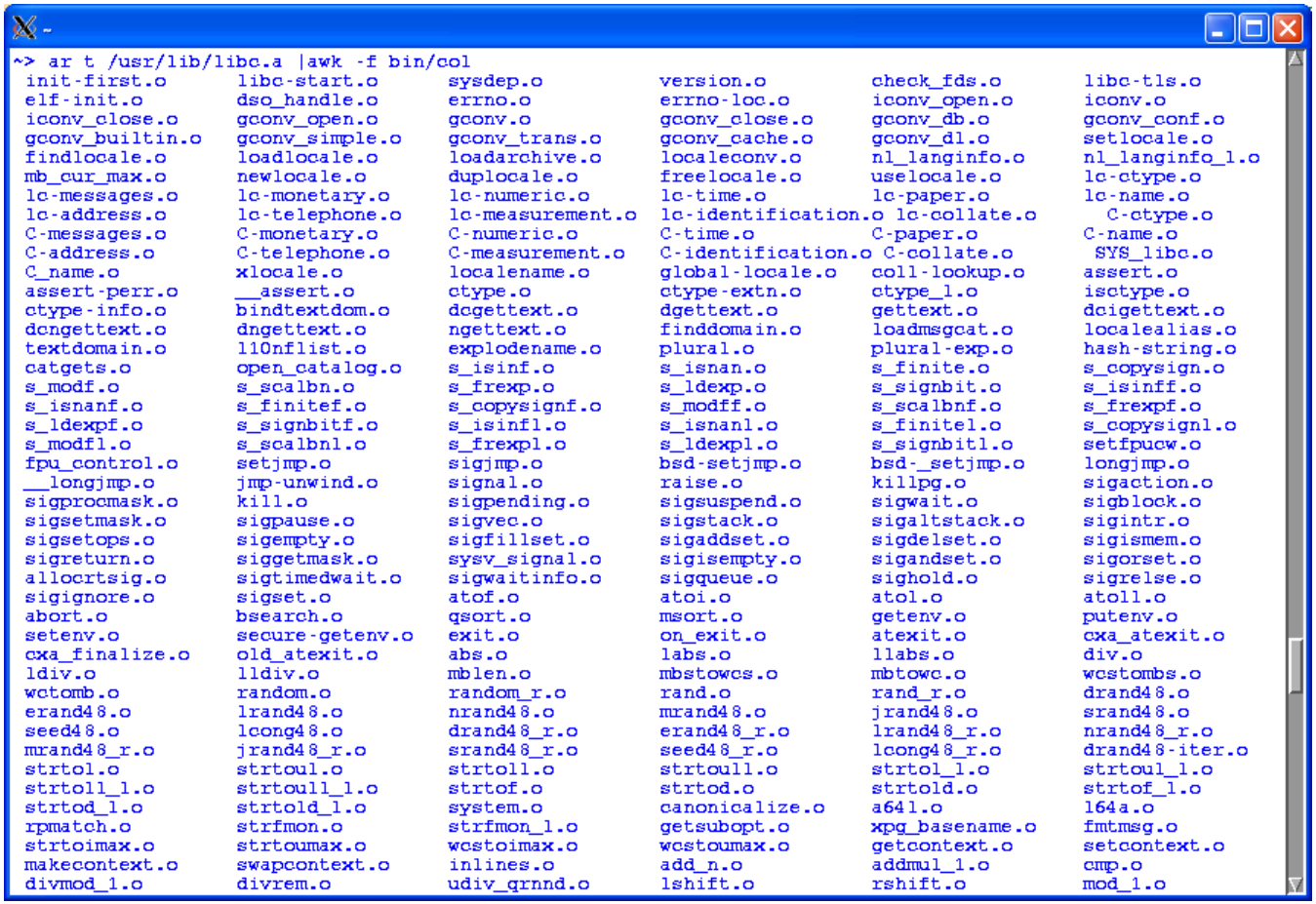
```
ar –t /usr/lib/libm.a |more
...
e_acos.o
e_acosh.o
e_asin.o
e_atan2.o
e_atanh.o
e_cosh.o
e_exp.o
e_fmod.o
e_log10.o
e_pow.o
...
```

Le programme GNU Archiver : `ar`

- crée,
- modifie
- et extrait des fichiers d'une archive.

Consultation de l'archive : **ar t**

`ar t` permet de consulter la liste des fichiers objets contenus dans l'archive :



```
~> ar t /usr/lib/libc.a |awk -f bin/col
init-first.o      libc-start.o      sysdep.o          version.o         check_fds.o       libc-tls.o
elf-init.o        dso_handle.o      errno.o           errno-loc.o       iconv_open.o      iconv.o
iconv_close.o     gconv_open.o      gconv.o           gconv_close.o     gconv_db.o        gconv_conf.o
gconv_builtin.o   gconv_simple.o    gconv_trans.o     gconv_cache.o     gconv_dl.o        setlocale.o
findlocale.o      loadlocale.o      loadarchive.o     localeconv.o      nl_langinfo.o     nl_langinfo_l.o
mb_cur_max.o      newlocale.o       duplocale.o       freelocale.o      uselocale.o       lc-ctype.o
lc-messages.o     lc-monetary.o     lc-numeric.o      lc-time.o         lc-paper.o        lc-name.o
lc-address.o      lc-telephone.o    lc-measurement.o  lc-identification.o lc-collate.o      C-ctype.o
C-messages.o      C-monetary.o      C-numeric.o       C-time.o          C-paper.o         C-name.o
C-address.o       C-telephone.o     C-measurement.o   C-identification.o C-collate.o       SYS_libc.o
C_name.o          xlocale.o         localename.o      global-locale.o   coll-lookup.o     assert.o
assert-perr.o     __assert.o        ctype.o           ctype-extn.o      ctype_l.o         isctype.o
ctype-info.o      bindtextdom.o     dcgettext.o       dgettext.o        gettext.o         dcigettext.o
dcngettext.o      dngettext.o       ngettext.o        finddomain.o      loadmsgcat.o      localealias.o
textdomain.o      l10nflist.o       explodename.o     plural.o          plural-exp.o      hash-string.o
catgets.o         open_catalog.o    s_isinf.o         s_isnan.o         s_finite.o        s_copysign.o
s_modf.o          s_scalbn.o        s_frexp.o         s_ldexp.o         s_signbit.o       s_isinff.o
s_isnanf.o        s_finitef.o       s_copysignf.o     s_modff.o         s_scalbnf.o       s_frexpf.o
s_ldexpf.o        s_signbitf.o      s_isinfl.o        s_isnanl.o        s_finitel.o       s_copysignl.o
s_modfl.o         s_scalbnl.o       s_frexpl.o        s_ldexpl.o        s_signbitl.o      setfpucw.o
fpu_control.o     setjmp.o          sigjmp.o          bsd-setjmp.o      bsd-_setjmp.o     longjmp.o
__longjmp.o       jmp-unwind.o      signal.o          raise.o           killpg.o          sigaction.o
sigprocmask.o     kill.o            sigpending.o      sigsuspend.o      sigwait.o         sigblock.o
sigsetmask.o      sigpause.o        sigvec.o          sigstack.o        sigaltstack.o     sigintr.o
sigsetops.o       sigempty.o        sigfillset.o      sigaddset.o       sigdelset.o       sigismem.o
sigreturn.o       siggetmask.o      sysv_signal.o     sigisempty.o      sigandset.o       sigorset.o
allocrtsig.o      sigtimedwait.o    sigwaitinfo.o     sigqueue.o        sighold.o         sigrelse.o
sigignore.o       sigset.o          atof.o            atoi.o            atol.o            atoll.o
abort.o           bsearch.o         qsort.o           msort.o           getenv.o          putenv.o
setenv.o          secure-getenv.o   exit.o            on_exit.o         atexit.o          cxa_atexit.o
cxa_finalize.o    old_atexit.o      abs.o             labs.o            llabs.o           div.o
ldiv.o            lldiv.o           mblen.o           mbstowcs.o        mbtowc.o          wcstombs.o
wctomb.o          random.o          random_r.o        rand.o            rand_r.o          drand48.o
erand48.o         lrand48.o         nrand48.o         mrand48.o         jrand48.o         srand48.o
seed48.o          lcong48.o         drand48_r.o       erand48_r.o       lrand48_r.o       nrand48_r.o
mrand48_r.o       jrand48_r.o       srand48_r.o       seed48_r.o        lcong48_r.o       drand48-iter.o
strtol.o          strtoul.o         strtoll.o         strtoull.o        strtol_l.o        strtoul_l.o
strtoll_l.o       strtoull_l.o      strtof.o          strtod.o          strtold.o         strtof_l.o
strtod_l.o        strtold_l.o       system.o          canonicalize.o    a64l.o            164a.o
rpmatch.o         strfmon.o         strfmon_l.o       getsubopt.o       xpg_basename.o    fmtmsg.o
strtoimax.o       strtoumax.o       wcstoimax.o       wcstoumax.o       getcontext.o      setcontext.o
makecontext.o     swapcontext.o     inlines.o         add_n.o           addmul_1.o        cmp.o
divmod_1.o        divrem.o          udiv_qrnnd.o      lshift.o          rshift.o          mod_1.o
```

Table des symboles de l'archive

Un fichier d'archive débute "normalement" par une table des symboles : l'**archive index**.

```
1   ~> nm -s /usr/lib/libc.a |more
2   Archive index:
3   ...
4   __fprintf in fprintf.o
5   fprintf in fprintf.o
6   _IO_fprintf in fprintf.o
7   __printf in printf.o
8   printf in printf.o
9   _IO_printf in printf.o
10  __snprintf in snprintf.o
11  snprintf in snprintf.o
12  __sprintf in sprintf.o
13  sprintf in sprintf.o
14  ...
15  fprintf.o:
16  00000000 W _IO_fprintf
17  00000000 T __fprintf
18  00000000 T fprintf
19           U vfprintf
20  printf.o:
21  00000000 T _IO_printf
22  00000000 T __printf
23  00000000 T printf
24           U stdout
25           U vfprintf
26  ...
```

- Il s'agit d'une description des modules et identificateurs permettant à l'éditeur de liens de résoudre les références correspondantes sans avoir à parcourir toute la bibliothèque.
- Une archive disposant d'un tel index **accélère l'édition des liens** avec la bibliothèque, et permet aux sous programmes de cette dernière de **s'appeler mutuellement** quels que soient leurs emplacements respectifs dans l'archive.

Analyser l'archive : **nm**

`nm` permet d'obtenir la liste des symboles se trouvant dans les fichiers objets :

```
1  ~> nm -s /usr/lib/libc.a |more
2  Archive index:
3  ...
4  __fprintf in fprintf.o
5  fprintf in fprintf.o
6  _IO_fprintf in fprintf.o
7  __printf in printf.o
8  printf in printf.o
9  _IO_printf in printf.o
10 __snprintf in snprintf.o
11 snprintf in snprintf.o
12 __sprintf in sprintf.o
13 sprintf in sprintf.o
14 ...
15 fprintf.o:
16 00000000 W _IO_fprintf
17 00000000 T __fprintf
18 00000000 T fprintf
19          U vfprintf
20 printf.o:
21 00000000 T _IO_printf
22 00000000 T __printf
23 00000000 T printf
24          U stdout
25          U vfprintf
26 ...
```

① L'index indique pour chaque symbole le fichier dans lequel il est défini.

② Ensuite pour chaque fichier, on trouve l'adresse relative (i.e offset) de définition des fonctions :

- Le symbole **T** (text symbol, global) signifie que la fonction est définie dans le fichier.
- Le symbole **U** signifie que la fonction est indéfinie et qu'elle devra être résolue à l'édition de lien : elle est ailleurs !
- Le symbole **W** (weak symbol) signifie que le symbole est défini dans le fichier MAIS qu'il peut être surchagé (à l'édition de lien) par une autre définition.

Création

```c
/* convert1.c : conversion francs euros */
#include <stdio.h>

/* x francs = x/6.55957 euros */
void franc_euro(double franc){
  printf("\t%.4lf francs = %.4lf euros \n",
         franc, franc/6.55957);
}
```

```c
/* convert2.c : conversion euros francs */
#include <stdio.h>

/* x francs = x/6.55957 euros */
void euro_franc(double euro){
  printf("\t%.4lf euros = %.4lf francs \n",
         euro, euro*6.55957);
}
```

① Compiler les fichiers sources en fichiers objets (avec l'option -c).

```
gcc -c convert1.c
gcc -c convert2.c
```

② Créer la bibliothèque, en remplacant ou juste en concaténant (rapidement !) :

```
ar r libconvert.a convert1.o convert2.o
ar vq libconvert.a *.o
```

Création de la table des symboles

```
1   ~> nm -s ../Code/libconvert.a
2
3   Archive index:
4   __franc__euro in convert1.o
5   __euro__franc in convert2.o
6
7   convert1.o:
8   00000000 b .bss
9   00000000 d .data
10  00000000 r .rdata
11  00000000 t .text
12  00000000 T __franc__euro
13           U __printf
14
15  convert2.o:
16  00000000 b .bss
17  00000000 d .data
18  00000000 r .rdata
19  00000000 t .text
20  00000000 T __euro__franc
21           U __printf
```

`ar` crée quasi automatiquement un **index des symboles** définis dans les modules objets relogeables de l'archive quand vous spécifiez le modificateur **s**.

Devant chaque symbole (nom de fonction, nom de variable) figure son adresse relativement à la section (T ou r : on y reviendra) à laquelle il appartient.

- Une fois créé, cet index est mis à jour dans l'archive à chaque fois que ar modifie son contenu (sauf dans le cas d'une de mise à jour **q**).
- Si l'archive n'en possède pas, l'utilitaire `ranlib` peut être utilisée pour n'ajouter que cette table à l'archive existante.

Convention de nommage et utilisation

```
~> gcc  convert2sens.c outils.c -L . -lconvert -o convertir
~> ./convertir
entrez un nombre : 100
convertir en euros (0) / francs (1) ? 0
100.0000 francs = 15.2449 euros
entrez un nombre : 15
convertir en euros (0) / francs (1) ? 1
15.0000 euros = 98.3936 francs
entrez un nombre : 0
```

Remarquez :

① L'option `L` qui permet d'indiquer où chercher la bibliothèque,

② L'utilisation de la convention de nommage qui permet de nommer l'archive `lib`**`convert`**`.a` par `-lconvert` ,

③ Et la génération de l'exécutable aprés tout cela !

Déploiement : utilisation avec répertoires

Si on suppose que la variable PROJ a été définie :

```
gcc convert2sens.c outils.c -I $PROJ/include -L $PROJ/lib –lconvert –o convertir
```

① Les fichiers d'entête sont alors localisés grâce à l'option `I` .

Cette option opère que le fichier d'entête soit inclus :

- avec < f.h >
- ou avec "f.h"

② Les archives sont alors localisées grace à l'option `L`

Motivation

Comme pour la programmation dans le tas (où on souhaite pouvoir allouer une zone mémoire dont la taille serait connue au ”runtime”),

➢ il est légitime de souhaiter pouvoir rendre ”dynamique” l'intégration d'un code (i.e. ensemble d'instructions) à un processus.

Définition :

Une bibliothèque dynamique, nommée Dynamic Link Library (.dll) pour Windows et nommée **shared object** (.so) ou ”dynamic library” sous UNIX, est un fichier de bibliothèque logicielle **utilisé par un programme exécutable, mais n'en faisant pas partie.**

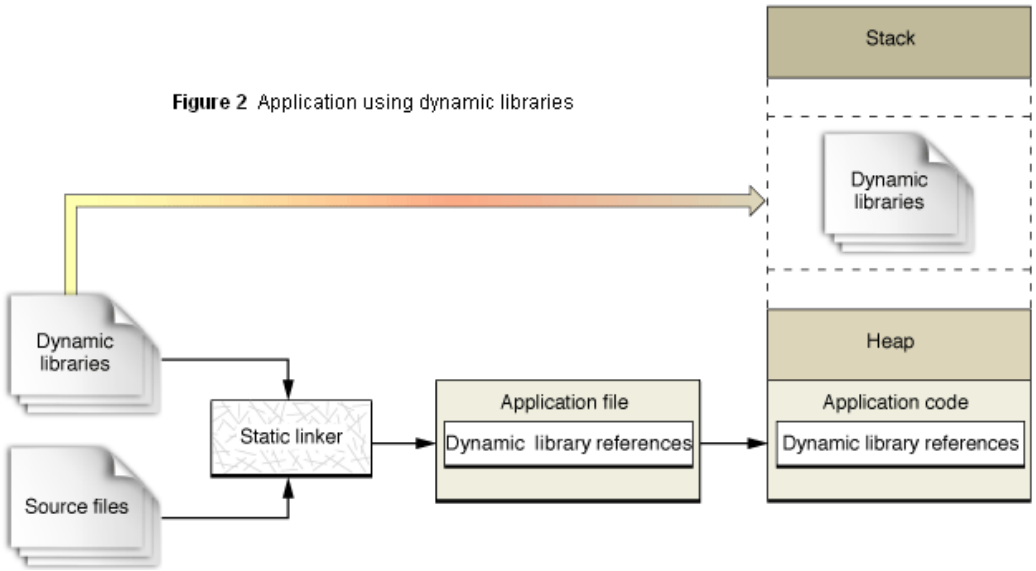
➠ Ce fichier contient des fonctions qui pourront être appelées pendant l'exécution d'un programme,

➠ sans que celles-ci soient incluses dans le fichier exécutable correspondant au programme.

Caractérisation

Le code de l'application ne contient pas le code de la bibliothèque.

- Il n'y a **que** "**des références à**" !



Le code de l'archive est chargée en mémoire et l'édition de lien à lieu :

- Soit au "launch time" : pendant le chargement en mémoire du programme,
- Soit au "run time" : pendant l'exécution du programme.

Pourquoi et Comment "Dynamique" ?

Pour comprendre, il faut analyser le code généré (`gcc`) :

```
1   #include <math.h>
2   int main(int argc, char * argv[]){
3     float x,y;
4     y = sin(x);
5   }
```

```
1   ad.out:     file format elf32-i386
2
3   Disassembly of section .plt:
4   080482b4 <__gmon_start__@plt-0x10>:
5    80482b4:       pushl  0x80495c0
6    80482ba:       jmp    *0x80495c4
7    80482c0:       add    %al,(%eax)
8     ...
9   080482e4 <sin@plt>:
10   80482e4:       jmp    *0x80495d0
11   80482ea:       push   0x10
12   80482ef:       jmp    80482b4 <_init+0x18>
13
14  Disassembly of section .text:
15  080483d4 <main>:
16    ...
17   80483e2:       sub    0x24,%esp
18   80483e5:       flds   0xfffffff4(%ebp)
19   80483e8:       fstpl  (%esp)
20   80483eb:       call   80482e4 <sin@plt>
21   80483f0:       fstps  0xfffffff8(%ebp)
22   80483f3:       add    0x24,%esp
23    ...
```

On constate la présence d'une PLT = Procedure Linkage Table

- ➟ Chaque appel de fonction (instruction **call**) passe par cette table pour trouver la fonction réellement appelée.
- ➟ Mécanisme de "redirection" ou adressage dynamique.

Concept de Code Relogeable

http://em386.blogspot.com/2006/10/resolving-elf-relocation-name-symbols.html

From the ELF 1.2 standard :

"Relocation is the process of connecting symbolic references with symbolic definitions."

- For example, when a program calls a function, the associated call instruction must transfer control to the proper destination address at execution.

In other words, relocatable files must have information that describes how to modify their section contents, thus allowing executable and shared object files to hold the right information for a process's program image. Relocation entries are these data.

- Ce mécanisme est rendu indispensable puisqu'on ne peut pas connaître, avant au minimum le lancement, la position en mémoire du code que l'on souhaite appeler.
- Le mécanisme de PLT permet de résoudre les adresses des fonctions à l'exécution.
- Un tel code est dit **relogeable** ou PIC (**Position Independant Code**).

Avantages/Inconvénients

No Good	Good
Code exécutable non ”self consistent”.	Réduction de la taille d’un exécutable, puisque certaines parties du logiciel se situent en dehors de lui,
Nécessite généralement la présence de plusieurs versions de la bibliothèques dans le système.	et la mise à jour possible des fonctions utiles pour toutes les applications qui les utilisent.
Complexité et Performances ?	Les applications bénéficient alors automatiquement des corrections ou des améliorations fonctionnelles de la bibliothèque.

Dynamisme et Partage

Dans le monde Unix, les bibliothèques dynamiques sont dénommées **shared** (i.e partagées).

- ➟ Il est vrai que le même fichier d'archive est partagé entre les différents programmes.

Mais il existe un autre lieu de partage !

Lorsque plusieurs programmes utilisent les mêmes bibliothèques, il est judicieux de ne charger celles-ci qu'une seule fois **en mémoire** et de laisser tous les programmes en utiliser la même copie.

- ➟ On parle alors de bibliothèques partagées.

Conclusion :

Une bibliothèque peut être dynamique sans pour autant être partagée, et partagée sans être dynamique.

Partage et Réentrance

Le fait de **partager des bibliothèques implique de fortes contraintes de conception** afin que les appels de fonction par un programme, susceptibles de modifier l'état interne de la bibliothèque, ne perturbent pas les autres programmes qui l'utilisent également.

Dans un système multitâche, comme on ne maîtrise pas les moments où les fonctions de la bibliothèque sont appelées, il est possible qu'une fonction soit appelée plusieurs fois simultanément.

- On appelle **réentrance** la faculté d'une bibliothèque à pouvoir être utilisée simultanément par plusieurs applications.
- On parle alors de **bibliothèque réentrante** ou même de **fonction réentrante** .

La première cause qui pourrait empêcher la réentrance c'est les effets de bords :

- pas de variables statiques donc pas de variables globales : CQFD !

Création d'une bibliothèque dynamique

En deux étapes :

① Compiler les fichiers objets au format PIC :

Option -fPIC :

```
gcc -fPIC -c convert1.c convert2.c $PROJ/include
```

② Créer la bibliothèque :

Option -shared :

```
gcc -shared -o libconvert.so convert1.o convert2.o
```

Utilisation problèmatique ?

Edition de liens et . . .

> gcc convert2sens.c outils.c -I $PROJ/include -L $PROJ/lib –lconvert –o convertir

> ./convertir

Run !

ld-linux.so : ./convertir : fatal : libconvert.so : open failed : No such file or directory

ou

./convertir : error while loading shared libraries : libconvert.so : cannot open shared object file : No such file or directory

Donc ça ne marche pas !

- Il semblerait que ”ld...” ne trouve pas le fichier `libconvert.so` ?
- Pourtant l'option `-L` est correctement positionnée !

Bibliothèques prérequises : `ldd`

Il existe un utilitaire `ldd` qui permet de trouver quelles bibliothèques partagées sont nécessaires au fonctionnement d'une application ou d'une autre bibliothèque partagée :

```
~/OnUtr/EnseignementsCurrent/Cours_EnvProgSys/Code> ldd /usr/lib/libGL.so.1
        linux-gate.so.1 =>  (0x00f7a000)
        libX11.so.6 => /usr/lib/libX11.so.6 (0x005c7000)
        libXext.so.6 => /usr/lib/libXext.so.6 (0x00712000)
        libXxf86vm.so.1 => /usr/lib/libXxf86vm.so.1 (0x06363000)
        libm.so.6 => /lib/libm.so.6 (0x004b7000)
        libpthread.so.0 => /lib/libpthread.so.0 (0x004e6000)
        libdl.so.2 => /lib/libdl.so.2 (0x004e0000)
        libdrm.so.2 => /usr/lib/libdrm.so.2 (0x00da6000)
        libc.so.6 => /lib/libc.so.6 (0x00371000)
        libXau.so.6 => /usr/lib/libXau.so.6 (0x0070d000)
        libXdmcp.so.6 => /usr/lib/libXdmcp.so.6 (0x005bf000)
        /lib/ld-linux.so.2 (0x00353000)
~/OnUtr/EnseignementsCurrent/Cours_EnvProgSys/Code> ldd ./convertir
        linux-gate.so.1 =>  (0x00a9b000)
        libconvert.so => not found
        libc.so.6 => /lib/libc.so.6 (0x00371000)
        /lib/ld-linux.so.2 (0x00353000)
```

- `ldd` demande à sa cible quelle sont ces dépendances en bibliothèques
- ensuite pour chacune, il demande au système d'exploitation où il va les trouver : Visiblement le système ne trouve pas `libconvert.so` !

Module de chargement : `ld.so`

On a vu qu'il y avait 2 "temps possibles" pour l'édition de liens dans le cadre d'une bibliothèque dynamique :

- au chargement du programme en mémoire : "launch time"
- à (durant) l'exécution : "run time"

Définition :

Le module logiciel du noyau responsable de l'édition de lien au chargement c'est le **chargeur** :

- /lib/ld.so (pour les formats a.out ou COFF)
- /lib/ld-linux.so (pour les formats ELF)

Mécanisme de recherche de `ld.so`

`ld.so` cherche les bibliothèques partagées dans les répertoires :

- `/lib` et `/usr/lib`
- ceux définis dans le fichier `/etc/ld.so.conf`
- ceux de la variable **LD_LIBRARY_PATH**

Résolution par **LD_LIBRARY_PATH** :

```
> ./convertir
./convertir : error while loading shared libraries : libconvert.so : cannot open shared object file : No such file or directory
> export LD_LIBRARY_PATH=. :$LD_LIBRARY_PATH
> ldd ./convertir
        linux-gate.so.1 => (0x0022b000)
        libconvert.so => ./libconvert.so (0x005de000)
        libc.so.6 => /lib/libc.so.6 (0x00371000)
        /lib/ldlinux.so.2 (0x00353000)
> ./convertir
entrez un nombre ...
```

Résolution statique à l'édition de liens.

L'option de compilation `-Wl,option` sert à passer une ou des options à l'éditeur de liens.

- `-Wl,-rpath` par exemple, pour qu'il inscrive dans l'application le chemin de l'archive.
- ainsi `-Wl,-rpath,dir` ajoute le répertoire `dir` à la liste du chemin de recherche des bibliothèques dans l'application.
- Attention pas d'espaces !

Résolution statique à l'édition de liens :

```
> gcc outils.c convert2sens.c -L . -lconvert -Wl,-rpath,/net1/home/menez/Code/ -o convertir2
> ldd ./convertir2
        linux-gate.so.1 => (0x0022b000)
        libconvert.so => /net1/home/menez/Code/libconvert.so (0x00113000)
        libc.so.6 => /lib/libc.so.6 (0x00371000)
        /lib/ldlinux.so.2 (0x00353000)
> ./convertir2
entrez un nombre ...
```

Versions de bibliothèques partagées

Les objectifs du concept de versions de bibliothèque sont :

- La gestion ”douce” de l’évolution des bibliothèques,
- et en conséquence, la gestion de la présence simultanée, dans le système, de différentes versions d’une même bibliothèque.

Mise en oeuvre :

Afin de permettre cela, dans le monde Unix, chaque bibliothèque a :

- un nom d’objet partagé ou `soname` ,
- un nom réel,
- un nom de lien.

Un scénario de gestion de version . . .

Supposons que l'on ait créé une bibliothèque $PROJ/lib/**libMyVa.so**

➢ Il s'agit du nom réel du fichier d'archive.

Supposons que l'on produise une nouvelle version de cette bibliothèque :

➢ $PROJ/lib/**libMyVb.so**

Une question se pose immédiatement :

Est ce que d'un point de vue d'une application utilisant cette bibliothèque, les deux versions sont compatibles ou pas ?

➢ A la vue du nom réel, elle sont différentes puisque dans deux fichiers différents :

Le nom réel n'aide pas !

➢ Pourtant peut être que la deuxième version ne modifie pas son interface et se contente de corriger quelques bugs ?

Elles seraient donc compatibles : Le nom réel n'aide vraiment pas !

SONAME

Le **SONAME** est un nom, une **information inscrite dans la bibliothèque**, qui va permettre de gérer la compatibilité des versions.

Pour voir le SONAME d'une bibliothèque :

```
> objdump -p /usr/lib/libGL.so.1.2 | grep SONAME
SONAME libGL.so.1
```

Mise en oeuvre :

Des versions de bibliothèques "compatibles" auront le même SONAME.

- Ainsi par exemple, /usr/lib/libGL.so.1.2 et /usr/lib/libGL.so.1.3 partageront sans doute le même SONAME :

  libGL.so.1

- D'ou la notion de "nom partagé" (**S**hared **O**bject **Name**) pour signifier leur compatibilité.

Le SONAME au niveau de l'exécutable

Une application compilée avec une version de bibliothèque **NE conserve PAS** l'information du nom réel de la bibliothèque qu'il devra faire charger au lancement,

➢ **Elle conserve son SONAME !**

Générer la bibliothèque avec le SONAME "toto" :

```
libdyn:
gcc -fPIC -c convert1.c convert2.c
gcc -shared -Wl,-soname,toto -o libconvert.so convert1.o convert2.o
```

Générer l'exécutable utilisant cette bibliothèque :

```
main :
gcc outils.c convert2sens.c -L . -lconvert  \
-Wl,-rpath,/net1/home/menez/Code/ -o convertir2
```

Trouver dans l'exécutable, le SONAME que l'on sait indispensable :

> objdump -p convertir2 | grep toto

NEEDED toto

ldd sur un exécutable

`ldd` sur un exécutable montre les SONAME :

```
> ldd convertir2
    linux-gate.so.1 => (0x00f16000)
    toto => not found
    libc.so.6 => /lib/libc.so.6 (0x00371000)
    /lib/ld-linux.so.2 (0x00353000)
```

Nom de lien

Définition :

Le nom de lien est le nom de la bibliothèque que le compilateur utilise à l'édition de lien.

Ceci permet de changer les fichiers réels sans modifier la commande de compilation.

- Aprés tout, sauf indication précise, l'édition de lien nécessite la présence de la bibliothèque et peut importe son nom de fichier réel si le SONAME est le bon.

Un exemple

```
libdyn : ctest.c ctest.h                    #Library
    gcc -Wall -fPIC -c ctest.c
    gcc -shared -Wl,-soname,libctest.so.1 -o libctest.so.1.0 *.o
    mv libctest.so.1.0 lib
    ln -sf libctest.so.1.0 lib/libctest.so
    ln -sf libctest.so.1.0 lib/libctest.so.1

main : main.c                               #Application
    gcc -Wall -Wl,-rpath,./lib/ -L lib main.c -o main -lctest
```

Le `nom réel` est : `libctest.so.1.0` et le SONAME est : `libctest.so.1`

① Le premier lien crée le `nom de lien` : `libctest.so`

➢ Sans lui l'abbréviation `-lctest` nécessaire à la fabrication de l'application serait inopérente !

② Le deuxième lien associe le SONAME au nom réel.

➢ Il est indispensable au chargeur qui devra, à partir du SONAME, retrouver le fichier réel correspondant.

Phase de chargement

Rôle du chargeur :

Au lancement, c'est le chargeur (ld-linux.so) qui devra associer à ce SONAME une bibliothèque "réelle".

Les étapes :

① On exécute l'application,

② Le chargeur de l'OS (ld-linux.so) analyse l'application pour connaître les bibliothèques partagées dont elle a besoin.

Ceci se fait au travers des SONAME !

③ Avec la liste SONAME ainsi obtenue, le chargeur consulte le cache (ld.so.cache) pour trouver les chemins des fichiers d'archive qu'il faut charger en mémoire.

④ ld-linux.so charge les fichiers réels en mémoire.

Conventions de nommage

En première approche on n'a pas pris le temps de décrire les conventions de nommages.

- MAIS elles sont VITALES ! (si on veut y comprendre quelque chose . . .)

Pour une bibliothèque dont la version statique s'appelerait `libMy.a` :

- Le SONAME est de la forme : `libMy.so.majeur`

  majeur étant le numéro de version *majeure*.

  Par exemple : `libMy.so.1`

- Le nom réel est de la forme : `libMy.so.majeur.mineur.version`

  Par exemple : `libMy.so.1.0`

- Le nom de lien est le SONAME sans numéro de version.

  Par exemple : `libMy.so`

Conventions de numérotation des versions

La première version a le numéro 1.0 ou 1.0.1 (major.minor.release)

① Si la modification est **rétro-compatible**, on augmente le numéro de version mineure !

- ➢ Nouvelle version avec des améliorations mais qui ne rajoute pas de nouvelles fonctions.
- ➢ Les modifications apportées n'affectant que le fonctionnement interne de la bibliothèque et le nombre de fonctions et de variables.
- ➢ Les paramètres et types de valeurs renvoyées pour les fonctions restant intacts.

② Si la modification est **incompatible** avec les versions antérieures, on augmente le numéro de version majeure.

- ➢ Suppression ou ajout de fonctions, modification de syntaxes d'appel de fonctions, . . .

③ Pour une version majeure ou mineure, une augmentation du dernier chiffre signifie des corrections de bogues, une compatibilité conservée et aucun changement significatif.

Installation de bibliothèques

En tant que root, on peut installer de nouvelles bibliothèques dans `/usr` ou `/usr/lib` .

A partir de `/etc/ld.so.conf` , `ldconfig` crée les liens symboliques nécessaires à la mise en place d'une nouvelle bibliothèque, et met à jour le contenu du cache (`/etc/ld.so.cache`) :

`ldconfig` :

```
> ldconfig -p | more
        895 libs found in cache '/etc/ld.so.cache'
        libz.so.1 (libc6) => /usr/lib/libz.so.1
        libz.so (libc6) => /usr/lib/libz.so
        libxslt.so.1 (libc6) => /usr/lib/libxslt.so.1
        libxslt.so (libc6) => /usr/lib/libxslt.so
        libxml2.so.2 (libc6) => /usr/lib/libxml2.so.2 ...
```

Lancer `ldconfig` à chaque fois qu'on installe une bibliothèque !

Mise en oeuvre

```
all: application
ofiles: convert1.c convert2.c
        gcc -fPIC -c convert1.c convert2.c
archivedyn : ofiles
        gcc -shared -Wl,-soname,libconvert.so.1 \
            -o lib/libconvert.so.1.0.1 convert1.o convert2.o
        ln -sf libconvert.so.1.0.1 lib/libconvert.so.1   # soname
        ln -sf libconvert.so.1 lib/libconvert.so         # link name
application : archivedyn outils.c convert2sens.c
        gcc outils.c convert2sens.c -Wl,-rpath,./lib/ -L ./lib/ \
           -o convertir -lconvert
```

```
~/OnUtr/EnseignementsCurrent/Cours_EnvProgSys/Code/TestLib> ls
total 56
drwxr-xr-x 3 systemes 4096 sep  2 15:43 .
drwxr-xr-x 4 systemes 4096 sep  2 15:20 ..
-rwxr--r-- 1 systemes  207 aoû 31 17:00 convert1.c
-rw------- 1 systemes 1204 sep  2 15:43 convert1.o
-rwxr--r-- 1 systemes  204 aoû 31 17:01 convert2.c
-rw------- 1 systemes 1204 sep  2 15:43 convert2.o
-rwxr--r-- 1 systemes  693 mar  3  2005 convert2sens.c
-rwx------ 1 systemes 6071 sep  2 15:43 convertir
-rwxr--r-- 1 systemes  235 mar  3  2005 convertir.h
drwxr-xr-x 2 systemes 4096 sep  2 15:43 lib
-rwxr--r-- 1 systemes  444 sep  2 15:43 Makefile
-rwxr--r-- 1 systemes  694 sep  2 15:21 outils.c
-rwxr--r-- 1 systemes  234 mar  3  2005 outils.h
~/OnUtr/EnseignementsCurrent/Cours_EnvProgSys/Code/TestLib> ls lib
total 16
drwxr-xr-x 2 systemes 4096 sep  2 15:43 .
drwxr-xr-x 3 systemes 4096 sep  2 15:43 ..
lrwxrwxrwx 1 systemes   15 sep  2 15:43 libconvert.so -> libconvert.so.1
lrwxrwxrwx 1 systemes   19 sep  2 15:43 libconvert.so.1 -> libconvert.so.1.0.1
-rwx------ 1 systemes 4458 sep  2 15:43 libconvert.so.1.0.1
```

Statique et/ou Dynamique : Choisir ?

Par défaut, Linux travaille avec des bibliothèques dynamiques.

- `gcc` produit du code relogeable,
- et l'édition de lien utilisera, si elle est présente, plutôt la version dynamique d'une archive.

Pour "piloter" plus finement ces choix, il faut utiliser des options de compilation permettant de renseigner l'éditeur de lien :

```
> gcc -static f.c
> gcc convertir.c -L ./lib/ -Wl,-Bstatic -lconverts -Wl,-Bdynamic -lconvertd
    -o convertir
```

`convertir` est donc une application utilisant deux bibliothèques :

- une statique : libconverts.a
- une dynamique : libconvertd.so

Chargement à l'exécution

Notion de plugin !

➠ On voit ça en Td.

Index :