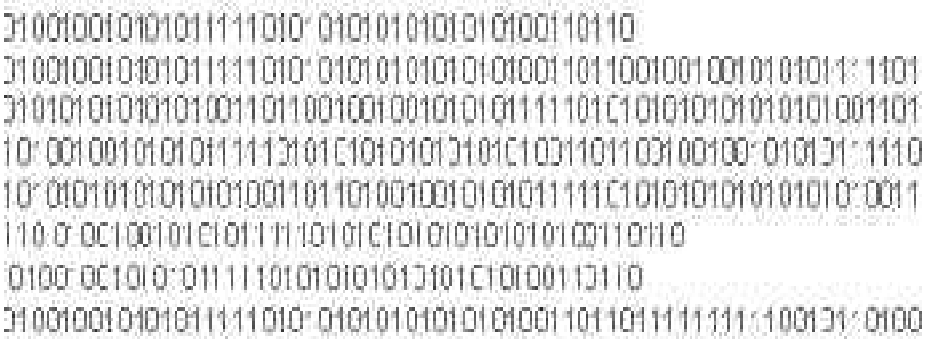
Rappels de développement



REF. OSP001

GAYET Thierry

Thierry.Gayet@laposte.net

Plan de la présentation

- Introduction
- Extensions de fichiers
- Chaîne de création
- Compilation
- Linkage
- Librairies dynamiques
- Librairie statique
- Conclusion
- Références

Introduction

A l'aide d'un langage donné, il est possible de compiler la retranscription d'un algorithme, lui- même issu d'une analyse, afin d'en obtenir un binaire, exécutable ou non, une librairie dynamique ou bien statique.

Une description de ces différentes transformations est détaillée dans ce document de synthèse.

Il traite principalement des plateformes SUN OS et Solaris, IBM Aix et bien entendu GUN Linux (toutes distributions).

Extensions 1/2

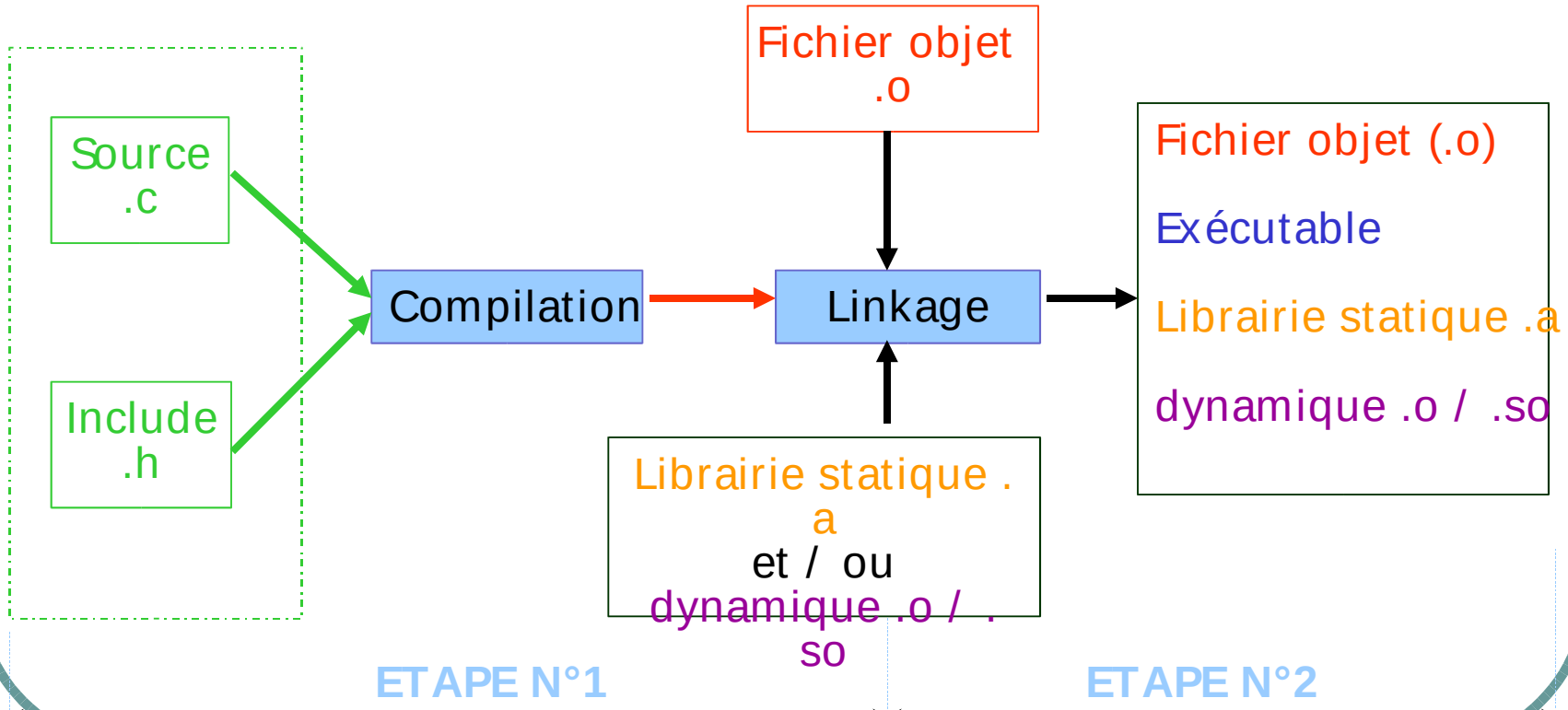
.c	Source en langage C nécessitant un pré-processing
.i	Source en langage C ne nécessitant pas de pré-processing
.cc .cp .cxx .cpp/ .c++ .C	Source en langage C++ nécessitant un pré-processing
.ii	Source en langage C++ ne nécessitant pas de pré-processing
.h	Fichier header ne devant pas être compilé ou linké !!!!

Extensions 2/2

.s	Source en langage assembleur nécessitant un pré-processing.
.S	Source en langage assembleur ne nécessitant pas de pré-processing.
.o	Fichier objet en langage machine.
.a	Librairie statique (et dynamique sous IBM Aix).
.so	Librairie dynamique (GNU Linux et SUN Solaris).

Chaîne de création...

Vue complète de la chaîne de compilation et de linkage :

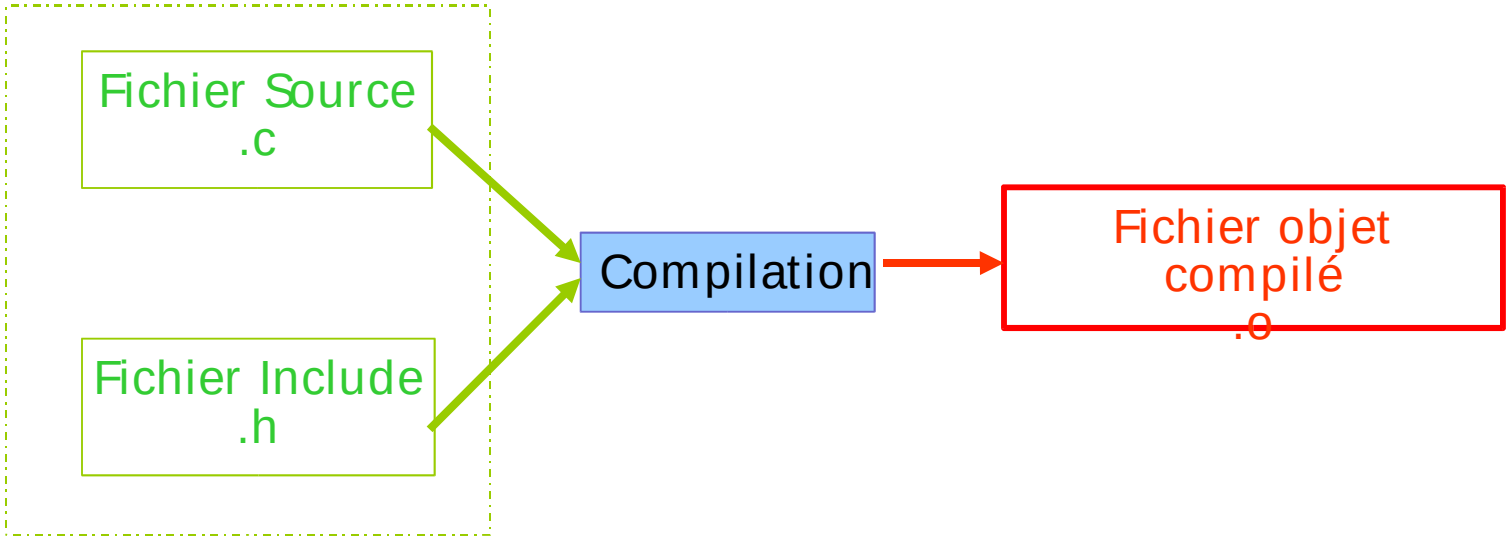


Etape n°1

LA COMPILATION

Etape 1 : la compilation 1/ 6

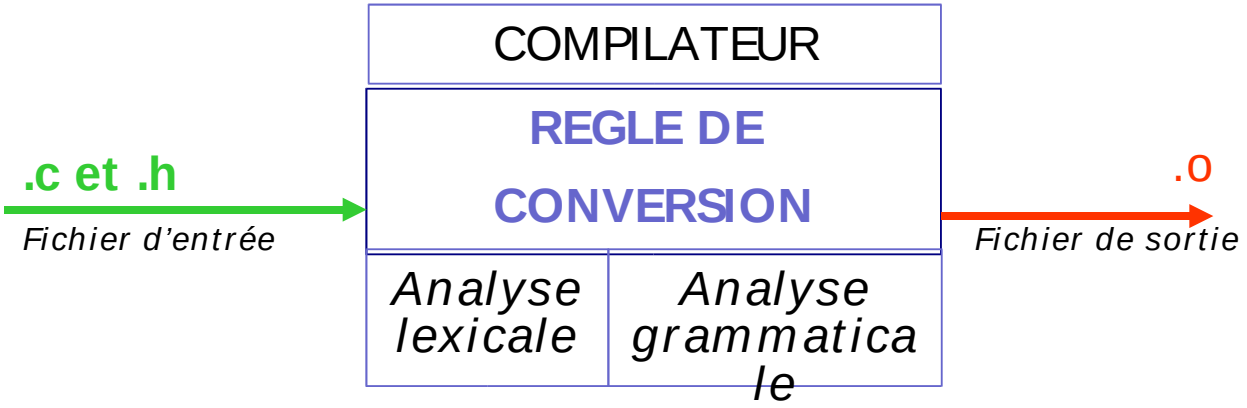
Analyse (lexicale Lex[(1)]/ Flex[(2)] et grammaticale Yacc[(1)]/ Bison[(2)]) du code source pour obtenir une version ciblée à une plateforme matérielle donnée (sparc, rs6000, x86, powerpc ...) :



(1) Outils issus des Unix commerciaux.
(2) Version libre (GNU) de ces outils.

Etape 1 : la compilation 2/6

Il s'agit d'une conversion de formats pouvant être représentée par la règle suivante :



Cette étape appelée « compilation » permet d'associer à un langage donné (C, Pascal, Java ...) une mnémonique objet liée à une plateforme matérielle donnée.

Ici comme dans la continuité de la chaîne, les extensions sont importantes pour le programme « *make* » car elle permettent de définir le bon programme à utiliser.

Etape 1 : la compilation 3/6

Exemple de compilation d'un seul source en langage C (**test.c**) et de son fichier header implicite (**test.h)** pour obtenir un fichier objet **test.o**

Chaîne d'appel au compilateur :

```
# gcc -c -o test.o test.c
```

Cette commande compile (-c) le source en langage C (.c) tout en générant en sortie un fichier objet (.o). L'extension de sortie est importante car dans le cas où elle est omise, le compilateur ajoutera la phase de linkage afin de générer un exécutable.

De plus, dans le cas où les fichiers sources (.c) et/ou les fichiers headers (.h) soient localisés dans un répertoire */src* respectivement */include,* différent du chemin courant, il est possible de préciser leur localisation au compilateur :

```
# gcc -c -o test.o ./src/test.c –I./include
```

Etape 1 : la compilation 4/6

Exemple de compilation dite modulaire de plusieurs sources en langage C (**test1.c et test2.c**) et d'un fichier header (**test.h)** pour obtenir un fichier objet **test.o**

Chaîne d'appel au compilateur :

```
# gcc -o test.o test1.c test2.c
```

A noter qu'il n'est pas possible d'utiliser le paramètre –c dans le cas d'une compilation modulaire (avec plusieurs fichiers sources).

Nom des compilateurs par plate-forme :

GNU Linux	gcc
SUN OS/Solaris	gcc ou acc
IBM Aix	gcc ou cc ou cc_r

Etape 1 : la compilation 5/6

Il est de même possible de rajouter quelques paramètres liés au débuggage :

```
# gcc -c -Wall -O2 –g -o test.o ./src/test.c –I./include
```

-Wall active tous les warning possibles

-g produit des informations de débuggage, utiles avec gdb, ddd, workshop, total view, ….

-01 Optimise du code lors de la phase de compilation (peut réduire la taille du code généré ainsi que le temps d'exécution).

-02 Optimisation avancée du code.

-Os active tous les types d'optimisation

(*Par défaut le compilateur n'optimise pas du tout le code*)

D'autres paramètres existent : -ggdb, -gstabs, etc …

A noter le paramètre –ansi qui force l'analyseur lexical du compilateur à n'utiliser que la sémantique propre au langage c ansi ou tout autre langage ISO C90 (C, fortran, etc …).

Etape 1 : la compilation 6/6

Dans certain cas il est parfois nécessaire de passer certains éléments au compilateur afin d'utiliser une compilation dite conditionnelle :

Exemple de compilation conditionnelle (.c) :

```
/* Code à inclure*/
#define CONDITION
printf("affichage de ce texte …");
#endif
```

Pour indiquer au compilateur qu'il doit inclure le printf dans le code source à compiler il faut passer à ce dernier le paramètre suivant :

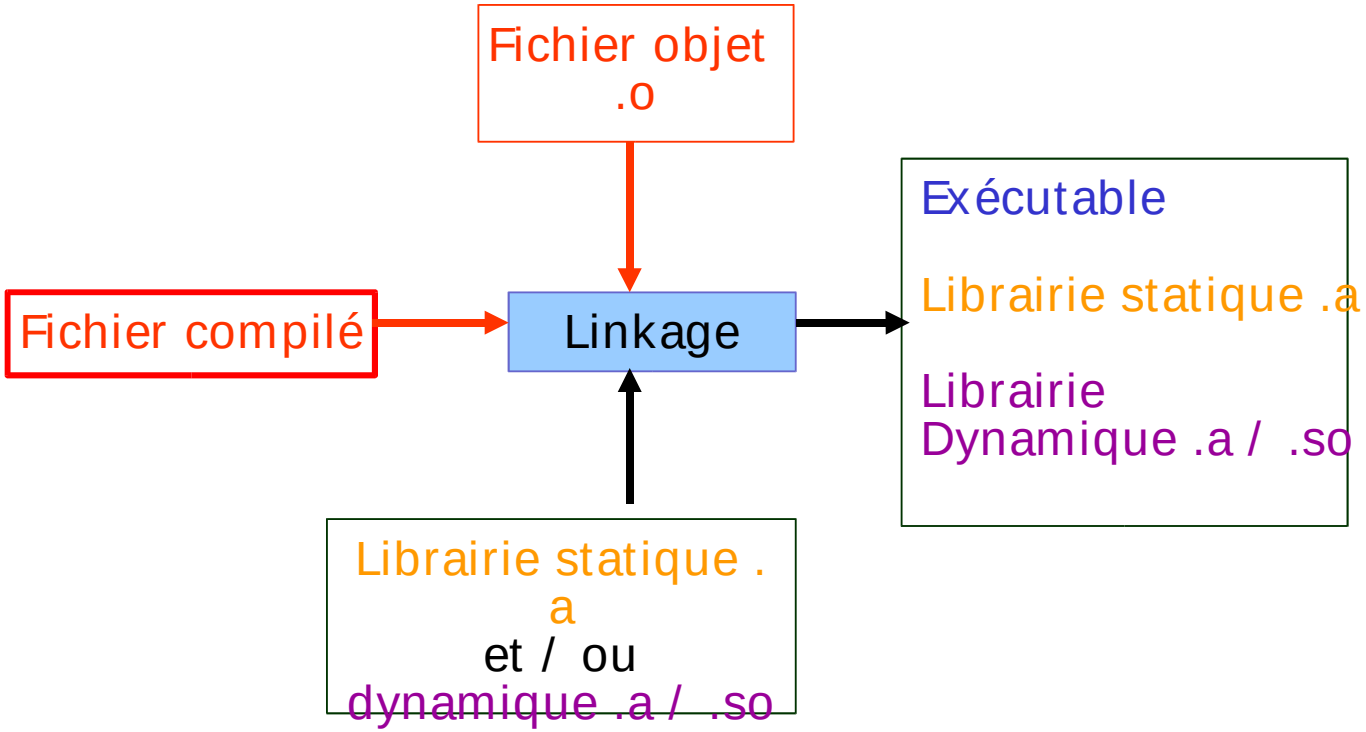
```
# gcc –c –o test.o –DCONDITION test.c
```

Etape n°2

LE LINKAGE

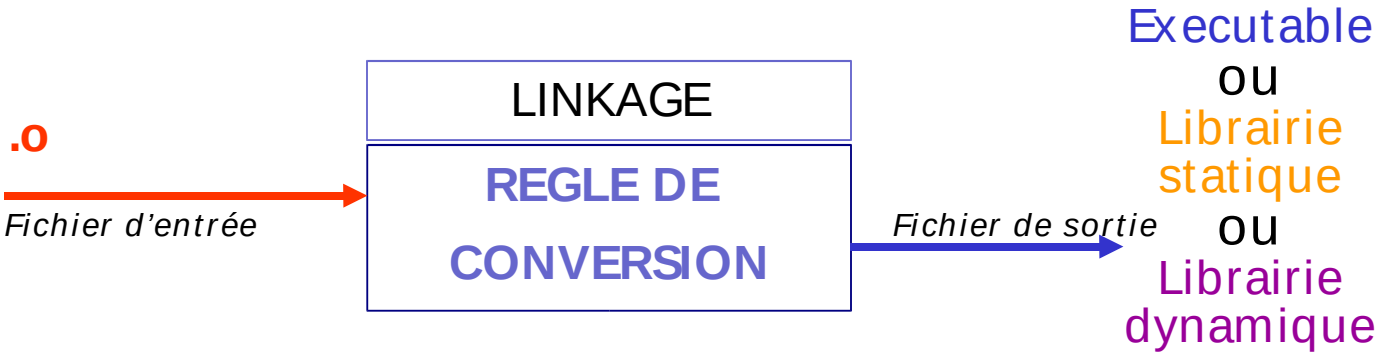
Etape 2 : le linkage 1/2

Représentation de la phase de mise à jour de la liste des symboles :



Etape 2 : le linkage 2/3

Il s'agit là aussi d'une conversion de format d'un fichier objet (.o) en un fichier exécutable.



Cette étape appelée « linkage » rajoute au fichier objet une table de symboles décrivant les différentes entrées.

Le programme utilisé habituellement pour réaliser cette conversion est « ld ». Certains compilateurs tels que « gcc ou cc » intègre aussi la fonction de linkage en natif.

Etape 2 : le linkage 3/3

Exemple : linkage d'un objet (.o) afin d'obtenir un exécutable :

```
# gcc –c –o test test.o
```

Commande identique à partir de plusieurs fichiers sources :

```
# gcc –o test test1.o test2.o
```

Nous pouvons remarquer que pour le linkage, les fichiers d'entrée sont des fichiers objets (.o).

Combinaisons possibles d'un fichier source (.c) :

```
# gcc –c –o test test.c
```
(*compilation & linkage*)

Pour information, sous le système GNU Linux le format par défaut est ELF et non a.out . La création d'un exécutable au lieu d'un objet indique au linkeur d'ajouter le point d'entrée main.

Editeurs de liens/ Linkeur par plate-forme :

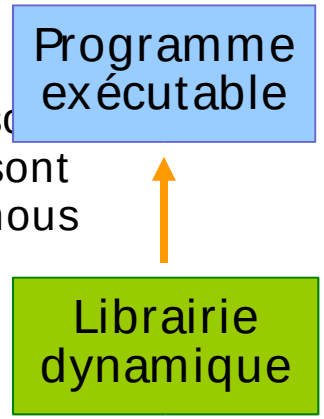
GNU Linux	gcc	ou ld
SUN OS/Solaris	gcc ou acc	ou ld
IBM Aix	gcc ou cc ou cc_r	ou ld

LES LIBRAIRIES DYNAMIQUES

Les librairies dynamiques 1/ 7

Définition :

Une librairie dynamique est une bibliothèque de fonctions qui so
chargées de façon dynamique. Plus précisément, ces librairies sont
chargées au moment de leur utilisation et déchargées lorsque nous
n'en n'avons plus besoin :



Avantages :

- taille des binaires et objets plus petits
- chargé une seule fois même si utilisé par plusieurs programme.
- permet de changer Dynamiquement de version de librairie

Désavantages :

- nécessite d'avoir la librairie dans le path.
- se doit d'être linkée par ld ou gcc.

Les librairies dynamiques 2/ 7

Création :

La création revient à linker un ensemble de fichier objets en un seul afin de former une bibliothèque. Néanmoins, même si le principe de base reste le même, les paramètres à passer diffèrent d'une plateforme à une autre :

```
GNU Linux          # gcc -c -shared -o test.so test.o
SUN OS/Solaris     # ld -dY -G      -o test.so test.o
IBM Aix            # ld -dY -G      -o test.o  test.o
```

D'autre part, même si sous SUN OS/ Solaris et IBM Aix, le linkeur utilisé de façon courante est ld, il est de plus de plus courant d'utiliser le compilateur même qui offre de nos jour cette double fonction de façon intrinsèque.

Extension d'une librairie dynamique :

```
GNU Linux                  .so
Sun OS / Sun Solaris       .so
IBM Aix                    .o
MICROSOFT Windows          .dll
```

Les librairies dynamiques 3/ 7

Test de dépendances :

Une fois compilé, il est possible de lister les dépendances d'un exécutable avec des librairies dynamiques (ainsi que leurs versions). Cela est utile pour permettre d'une part de savoir si un programme est compilé avec une librairie dynamique et d'autre part le nom de cette librairie.

```
# /usr/bin/ldd test
```

Exemple de résultat : # ldd test

```
libanasm7.so => /home/tgayet/vittam2/lib.i386_linux/libanasm7.so
                (0x40017000)
libc.so.6 => /lib/tls/libc.so.6 (0x42000000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Quand bien même, le programme n'utilise pas de librairie dynamique « utilisateur », nous pouvons remarquer qu'implicitement, ce dernier utilise des librairies dynamiques système comme celle de glibc : ***libc.so.6*** (implémentation des fonctions génériques utilisable en langage C).

Les librairies dynamiques 4/7

D'autre part, cette librairie possède la particularité de posséder un point d'entrée.
En effet, il est possible de l'exécuter tel un programme : `# /lib/libc.so.6`

```
GNU C Library stable release version 2.3.2, by Roland McGrath et al.
Copyright (C) 2003 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
Compiled by GNU CC version 3.2.2 20030222 (Red Hat Linux 3.2.2-5).
Compiled on a Linux 2.4.20 system on 2003-11-12.
Available extensions:
        GNU libio by Per Bothner
        crypt add-on version 2.1 by Michael Glad and others
        linuxthreads-0.10 by Xavier Leroy
        The C stubs add-on version 2.1.2.
        BIND-8.2.3-T5B
        NIS(YP)/NIS+ NSS modules 0.19 by Thorsten Kukuk
        Glibc-2.0 compatibility add-on by Cristian Gafton
        libthread_db work sponsored by Alpha Processor Inc
Thread-local storage support included.
Report bugs using the `glibcbug' script to <bugs@gnu.org>.
```

Une autre source d'information est :

`# /usr/bin/gcc -v`

```
Reading specs from /agl/tools/perl/current/bin/
../lib/gcc-lib/i386-redhat-linux/3.2.2/specs
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man
 --infodir=/usr/share/info
--enable-shared --enable-threads=posix --disable-checking
--with-system-zlib
--enable-__cxa_atexit --host=i386-redhat-linux
Thread model: posix
gcc version 3.2.2 20030222 (Red Hat Linux 3.2.2-5)
```

Les librairies dynamiques 5/7

PATH :

Lors de l'exécution d'un programme nécessitant une librairie dynamique,il peut être parfois nécessaire de modifier le cache système lié aux librairies dynamiques. En effet, si cette dernière ne se trouve pas dans dans le répertoire courant du binaire, une erreur de chargement se produira. Pour résoudre ce problème de chemin, il existe deux solutions :

1. Modifier la variable d'environnement **LD_LIBRARY_PATH** afin d'y ajouter le chemin souhaité :

```
setenv LD_LIBRARY_PATH $ LD_LIBRARY_PATH:/ nouveau_chemin
ou      export LD_LIBRARY_PATH= $ LD_LIBRARY_PATH:/ nouveau_chemin
```

nb : sous IBM Aix, la variable se nomme : LIBPATH

2. Réactualiser le cache lié aux librairies :

Fichier de paramétrage du cache : `/etc/ld.so.conf`
Commande de régénération : `/sbin/ldconfig` (1)

(1) *En root*

Il est courant de voir le fichier ld.so.conf contenir les chemins suivants : */lib, /usr/lib (parfois un alias), /usr/X11R6/lib, /usr/local/lib, /usr/openwin/lib, /opt/kde3/lib, /opt/gnome2/lib ...*

Les librairies dynamiques 6/ 7

Exemple ***détaillé*** d'utilisation en langage C du chargement d'une librairie dynamique :

test.c

```
#include <dlfcn.h>
#include <stdlib.h>
#include <stdio.h>

Typedef void (*test_fonction)(void);

int main(void)
{
    void *module;



    module        = dlopen("libtest", RTLD_LAZY);
    test_fonction = dlsyn(module, "fonction_test");

    /* utilisation de la fonction ici */

    dlclose();
}
```

Exemple de chargement de la fonction test dans la Librairie dynamique ***libtest.so***

Ouverture (handle) / Chargement de la librairie libtest.so

Recherche du pointeur de la fonction fonction_test()

Excution de la fonction

Fermeture de la librairie

A noter que le chargement est aujourd'hui Implicite et que ces étapes sont tranparentes. Les fonctions dlopen, dlsyn et dlclose sont implémentés dans la glibc.

Les librairies dynamiques 7/ 7

Linkage :

Une librairie dynamique n'est liée à un programme qu'au moment du linkage :

```
GNU Linux          # gcc -fPIC test.o –L./lib –ltest
SUN OS/Solaris     # gcc -KPIC test.o –L./lib –ltest
IBM Aix            # gcc       test.o –L./lib –ltest
```

*Nb : Le préfixe **lib** est rajouté automatiquement devant* **test** pour former **libtest.**

–fPIC définit que le bloc dynamique qui sera chargé en mémoire pourra être placé à différente zone.

Tout comme pour la phase de compilation il est possible de préciser certains paramètres. En effet, lors du linkage il est parfois nécessaire d'inclure des librairies statiques ou dynamiques. Le paramètre nécessaire pour ces faire est le suivant :

```
-l./lib –L./lib/test
```

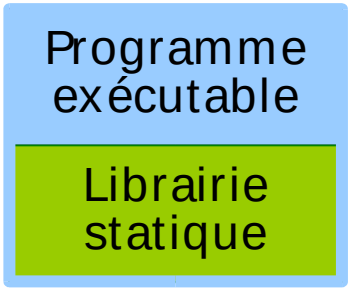
Le paramètre –l demande à l'éditeur de lien de rechercher dans le chemin ./lib, à partir du chemin courant, une bibliothèque statique libtest.a ou bien partagée/dynamique libtest.so. L'éditeur de liens parcourt tous les répertoires de bibliothèques standards (cf variable d'environnement LD_LIBRARY_PATH) et tous les -L indiqués. Si le linkeur trouve à la fois une librairie statique et dynamique, il choisit de préférence la librairie dynamique sauf si le paramètre -static est passé aussi.

LES LIBRAIRIES STATIQUES

Les librairies statiques 1/4

Définition :

Une librairie statique est comme la librairie dynamique, une bibliothèque de fonctions mais au lieu que le code soit dissocié du programme et chargé au moment de son utilisation propre, ce dernier est compilé en dur.



Avantages :

- facile à gérer.
- ne nécessite rien d'autre ; facilement diffusable.

Désavantages :

- Peut former des binaires de taille conséquente ;
  augmente donc la consommation de ressources, mais une fois alloué en RAM, aucun autre
  chargement n'est nécessaire.

Les librairies statiques 2/4

Création :

La création d'une librairie statique requiert l'utilisation du programme ar (*commande uniforme quelque soit la plateforme*) :

```
# /usr/bin/ar –rv test.a test1.o test2.o test3.o
ou
# /usr/bin/ar –rv test.a *.o
```

Cela rassemble les 3 fichiers objets (test1.o, test2.o et test3.o) en une librairie statique (test.a). L'extension .a d'une librairie statique est constante quelque soir la plateforme Unix à l'exception de MS Windows. (**.lib**).

Extension d'une librairie statique :

GNU Linux	.a
Sun OS / Sun Solaris	.a
IBM Aix	.a
MICROSOFT Windows	.lib

Il est possible de solliciter la recréation de cette table (utile pour SUN OS mais implicite sur les autres plateformes) :

```
# ranlib test.a (1)(1) Implicite sur une majorité de plateforme sauf sous SUN OS.
```

Les librairies statiques 3/ 4

Test :

Il est aussi possible de lister le contenu des fichiers objets inclus dans la librairie :

```
# /usr/bin/ar –t test.a
```

Exemple de résultats :

```
ad_server.o
cominter.o
com_util.o
filointer.o
miscell.o
paral.o
pilot.o
simul_api.o
spyinter.o
userint.o
```

La commande suivante permet de lister cette table de symboles :

```
# /usr/bin/nm  test.a
```

Légende :

U : undefined (implémentation externe)
T : implémenté à l'intérieur

Exemple de résultats :

```
ad_server.o:
00000099 T affichage_etat_client
00000004 C bDebugAd
00000004 C bDebugSu
         U bTrace
00000390 T close_socket
0000040f T close_tab_client
0000046d T connexion_client

cominter.o:
         U atoi
00000004 d bComInit
00000000 d bNoInitWarn
00000010 b bTrComInter
```

« Pour un objet, cela liste les fonctions et pour une librairie les fonctions par objet. »

Les librairies statiques 4/4

Linkage :

Le linkage d'un binaire avec une librairie statique permet d'ajouter les fonctions implémentées
à l'intérieur de cette dernière. Au final, la table de symboles ainsi générée sera commune.

Compilation d'un fichier source (test.c) avec la librairie statique libtest.a afin de générer le binaire (test) :

```
GNU Linux       # gcc -c test.o libtest.a -o test
SUN Solaris     # gcc -c test.o libtest.a -o test
IBM Aix          # gcc -c test.o libtest.a -o test
```

Le linkage d'une librairie statique avec un binaire est identique au linkage d'un fichier objet.

Conclusion

Jespère que ce document vous rendra service.

Pour davantage d'aide sur les différentes commandes :

```
# man commande

# commande --help

Les how to sur la compilation (glibc, gcc, gdb …).

Les forums dédiés (Newsgroups).

les sites dédiés.

Et bien entendu Google.fr

http://www.fortran-2000.org/ArnaudRecipes/sharedli.html
http://www.nyangau.fsnet.co.uk/dll/dll.htm
```

Making of...

Diaporama réalisé à partir des deux projets open source suivant :

 OpenOffice.fr

 Gimp

Fin

Bonne programmation