CM 16 - BIBLIOTHÈQUES

G. Bianchi, G. Blin, A. Bugeau, S. Gueorguieva, R. Uricaru

2015-2016

Programmation 1 - uf-info.ue.prog1@diff.u-bordeaux.fr

université de BORDEAUX

UTILISER UNE BIBLIOTHÈQUE

▷ Pour utiliser une bibliothèque, il est nécessaire d'inclure le(s) fichier(s) d'en-tête correspondant

▷ Sans cela, les symboles définis dans la bibliothèque ne sont pas utilisables

▷ Il faut également indiquer au compilateur qu'il doit utiliser la bibliothèque correspondante en chargeant un fichier **.a** ou **.so**

▷ L'option `-l` de **gcc** permet de spécifier le nom du fichier bibliothèque à rechercher et charger

▷ `gcc ... -lfoo` demande la recherche et le chargement de la librairie **libfoo.a** ou **libfoo.so**

▷ Si le fichier n'est pas dans le répertoire **/usr/lib**, il faut préciser au compilateur son chemin avec `-Lchemin`

BIBLIOTHÈQUE STATIQUE

BIBLIOTHÈQUE STATIQUE

▷ Une bibliothèque statique est représentée par un fichier **.a** contenant un ou plusieurs fichier(s) **.o**

▷ A **la compilation**, les portions de code nécessaires sont copiées dans l'exécutable

▷ Intérêt: l'exécutable est auto-suffisant (plus besoin de la bibliothèque)

▷ Désavantage: une redondance de code entre les exécutables (MAJ de la librairies sans effet)

▷ Elles sont peu utilisées (essentiellement historique)

EXEMPLE

▷ Bibliothèque (**utf8.c**/**utf8.h**) permettant la lecture et l'écriture d'un caractère en UTF8

```c
#include <stdio.h>
#include <stdint.h>

typedef uint16_t unichar;

/* This function writes a 2-bytes unicode character in the given
* file encoding it in UTF8. Returns 0 if an error occurs; 1
* otherwise. */
int fputc_utf8(unichar c,FILE* stream);

/* Reads an UTF8 encoded character from the given file and returns
* its unicode number. Returns EOF if the end of file has been
* reached. Prints an error and returns '?' if the end of file is
* found while reading a compound character, or if there is an
* encoding error. */
int fgetc_utf8(FILE* stream);
```

EXEMPLE

- Etape 1: créer le fichier objet **.o**

```
$>gcc -c utf8.c
```

- Etape 2: créer la bibliothèque **.a**

```
$>ar rs libutf8.a utf8.o
```

- Etape 3: visualiser son contenu

```
$>nm --defined-only libutf8.a

utf8.o:
00000157 T fgetc_utf8
00000000 T fputc_utf8
```

EXEMPLE

▷ **test.c** devrait afficher ”é” (code hexa E9), Ã© si le terminal n’est pas en UTF8

```
#include <stdio.h>
#include "utf8.h"

int main(int argc, char* argv[]){
  fputc_utf8(0xE9,stdout);
  fputc_utf8('\n',stdout);
  return EXIT_SUCCESS;
 }
```

▷ La librairie **libutf8.a** est présente dans le répertoire courant

```
$>gcc -std=c99 test.c -L. -lutf8
$>./a.out
Ã©
```

BIBLIOTHÈQUE PARTAGÉE

BIBLIOTHÈQUE PARTAGÉE

▷ Une bibliothèque partagée est représentée par un fichier **.so** (**s**hared **o**bject)

▷ A **l'exécution**, l'éditeur de liens dynamique ira chercher le code dans le fichier **.so**

▷ Intérêts:

▷ Si plusieurs programmes partagent l'utilisation de la bibliothèque, cette dernière n'est présente qu'une fois en mémoire

▷ En cas de mise à jour de la bibliothèque, les exécutables en profitent automatiquement

EXEMPLE

▷ Etape 1: créer le fichier objet **.o** avec l'option **-fPIC** (**P**osition **I**ndependent **C**ode)

```
$>gcc -fPIC -c utf8.c
```

▷ Etape 2: créer la bibliothèque avec l'option **-shared**

```
$>gcc -shared -o libutf8.so utf8.o
```

▷ Etape 3: créer l'exécutable

```
$>gcc test.c -L. -lutf8
```

EXEMPLE

▷ Etape 4: l'exécution ne marche pas

```
$>./a.out
./a.out: error while loading shared libraries: libutf8.so:
cannot open shared object file: No such file or directory
```

▷ Etape 5: l'outil **ldd** fournit les explications en affichant les dépendances

```
$>ldd a.out
    linux-gate.so.1 =>  (0xffffe000)
    libutf8.so => not found
    libc.so.6 => /lib/tls/libc.so.6 (0xb7e8f000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0xb7fea000)
```

▷ Le fichier n'étant pas dans **/usr/lib**, l'éditeur dynamique ne sait pas où le chercher

SOLUTIONS

▷ Copier le fichier dans **/usr/lib**

▷ Si on n'a pas accès à **/usr/lib**, on doit compiler l'exécutable avec l'option **-Wl, -rpath,chemin_du_so**

```
$>gcc test.c -L. -lutf8 -Wl,-rpath,.
$>ldd a.out
    linux-gate.so.1 =>  (0xffffe000)
    libutf8.so => ./libutf8.so (0xb7fe6000)
    libc.so.6 => /lib/tls/libc.so.6 (0xb7e8f000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0xb7fea000)
$>./a.out
Ã©
```

CONVENTION DE NOMMAGE

▷ Le nom d'une librairie suit une convention de nommage utilisant les notions de linker name, soname et real name

▷ Le **soname** est composé du préfixe lib suivi du nom de la librairie, de l'extension .so et d'un **numéro de version** précédé d'un .

  ▷ e.g., libutf8.so.1

  ▷ Le numéro de version est incrémenté à chaque changement d'interface de la librairie (i.e., le .h)

CONVENTION DE NOMMAGE

▷ On parle de **fully-qualified soname** lorsque le nom inclut comme préfixe le chemin de stockage de la librairie et correspond à un lien symbolique vers le **real name** de la librairie

▷ Le **real name** de la librairie correspond au ”vrai” nom de la librarie

▷ En ajoutant au **soname**, un **numéro de version mineure** (précédé d’un **.**) suivi d’un **numéro de release** (précédé d’un **.**) [ce dernier est optionnel]

▷ e.g., libutf8.so.1.0.2

▷ Le **linker name** de la librairie correspond au nom utilisé par le compilateur pour l’édition des liens et correspond au **soname** sans information de version

▷ e.g., libutf8.so

CONVENTION DE NOMMAGE

▷ L'intérêt de cet ensemble de nommages est de permettre la mise à jour des librairies sans impact sur les programmes l'utilisant

▷ Pour ce faire, les programmes ne manipulent que les **soname**

```
$>ldd a.out
    linux-gate.so.1 =>  (0xffffe000)
    libutf8.so => ./libutf8.so (0xb7fe6000)
    libc.so.6 => /lib/tls/libc.so.6 (0xb7e8f000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0xb7fea000)
```

CONVENTION DE NOMMAGE

▷ C'est un programme du système qui gère la mise à jour des librairies : ldd

▷ Tout est géré sous la forme de liens symboliques

```
$>ls -l /usr/lib/libm.so
lrwxrwxrwx 1 root root 19 jun 23 2014 /usr/lib/libm.so->../../lib/
libm.so.6

$>ls -l /lib/libm.so.6
lrwxrwxrwx 1 root root 13 jun 23 2014 /lib/libm.so.6->libm-2.3.3.so

$>ls -l /lib/libm-2.3.3.so
-rwxr-xr-x 1 root root 141380 jun 23 2004 /lib/libm-2.3.3.so
```

BIBLIOTHÈQUE DYNAMIQUE

▷ Les bibliothèques dynamiques sont chargées à l'exécution du programme (DL – Dynamically Loaded Libraries)

▷ Cela s'effectue à l'aide de fonctions définies dans **dlfcn.h**

▷ Pour inclure la librairie **libdl.so**, il faut compiler le programme avec l'option **-ldl**

▷ Ceci donne accès à plusieurs fonctions:

```
void* dlopen(const char* filename, int flag);
int dlclose(void* handle);
void* dlsym(void* handle,char* symbol);
const char* dlerror(void);
```

BIBLIOTHÈQUE DYNAMIQUE - CHARGEMENT

▷ La fonction `void* dlopen(const char* filename, int flag);`

▷ Charge la librairie dynamique indiquée par le paramètre `const char* filename` et retourne un pointeur la désignant ou `NULL` en cas d'erreur

▷ Le paramètre `int flag` permet de contrôler le degré de résolution des symboles de la librairies

▷ La constante **RTLD_LAZY** demande la résolution des symboles quand nécessaire

▷ La constante **RTLD_NOW** demande la résolution de tous les symboles utilisés dans la librairie

BIBLIOTHÈQUE DYNAMIQUE - DÉCHARGEMENT / RECHERCHE

▷ La fonction `int dlclose(void* handle);`

▷ Décharge la librairie dynamique indiquée si elle n'est plus utilisée par aucun programme

▷ La fonction `void* dlsym(void* handle, char* symbol);`

▷ Cherche le symbole indiqué par le paramètre `char* symbol` dans la librairie dynamique indiquée et retourne un pointeur le désignant ou `NULL` si le symbol n'est pas trouvé

▷ La fonction `const char* dlerror(void);`

▷ Retourne un pointeur sur une chaîne décrivant la dernière erreur qui s'est produite, ou `NULL` si la dernière erreur a déjà été gérée par un appel à la fonction `dlerror`

BIBLIOTHÈQUE DYNAMIQUE - EXEMPLE

▷ Nous proposons de mettre en oeuvre une bibliothèque simpliste **helloWorld**

hwFR.c

```
#include <stdio.h>
void hello(){
    printf("Bonjour monde!\n");
}
```

hwEN.c

```
#include <stdio.h>
void hello(){
    printf("Hello world!\n");
}
```

▷ Construction des libraires dynamiques

```
$>gcc -fPIC -c hwFR.c
$>gcc -shared -o libhwFR.so hwFR.o
$>gcc -fPIC -c hwEN.c
$>gcc -shared -o libhwEN.so hwEN.o
```

BIBLIOTHÈQUE DYNAMIQUE - EXEMPLE

```c
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
int main(int argc, char** argv){
  void* dl;
  void (*foo)(void);
  dl=dlopen(argv[1],RTLD_LAZY);
  if(dl==NULL){
    fprintf(stderr,"%s\n",dlerror());
    exit(EXIT_FAILURE);
  }
  foo=dlsym(dl,argv[2]);
  if(foo==NULL){
    fprintf(stderr,"%s\n",dlerror());
    exit(EXIT_FAILURE);
  }
  foo();
  dlclose(dl);
  return EXIT_SUCCESS;
}
```

```
$> gcc -std=c99 test.c -ldl
$> ./a.out ./libhwFR.so hello
Bonjour monde!
$> ./a.out ./libhwFR.so ello
dlsym(0x7fc2da403240, ello):
symbol not found
$> ./a.out ./libhwEN.so hello
Hello world!
$> ./a.out ./libhwDE.so hello
dlopen(./libhwDE.so, 1): image
not found
```

BIBLIOTHÈQUES ET VISIBILITÉ

- ⚠ Même si un élément n'est pas déclaré dans un fichier d'en-tête, il est accessible
  - Par exemple, le symbole **hello** n'était pas déclaré
- Pour rendre un élément non visible, il faut interdire l'export de son symbole en le déclarant avec **static**

hwFR.c

```
#include <stdio.h>
void hello(){
    printf("Bonjour monde!\n");
}
static void bonjour(){
    printf("Bonjour monde!\n");
}
```

```
$>gcc -fPIC -c hwFR.c
$>gcc -shared -o libhwFR.so hwFR.o
$> nm libhwFR.so
00000f60 T _helloWorld
         U _printf
         U dyld_stub_binder
```

BIBLIOTHÈQUES ET VISIBILITÉ

hwEN.c

```c
#include <stdio.h>
void hello(){
    printf("Hello world!\n");
}
void bonjour(){
    printf("Hello world!\n");
}
```

```
$> ./a.out ./libhwFR.so hello
Bonjour monde!
$> ./a.out ./libhwFR.so bonjour
dlsym(0x7fc2da403240, bonjour):
symbol not found
$> nm libhwEN.so
00000f60 T _bonjour
00000f30 T _helloWorld
         U _printf
         U dyld_stub_binder
$> ./a.out ./libhwEN.so hello
Hello world!
$> ./a.out ./libhwEN.so bonjour
Hello world!
```

PACKAGING SELON DEBIAN/UBUNTU

PRINCIPE

▷ Les applications/bibliothèques sont packagées sous forme binaire et/ou sources dans des repositories

▷ Le système de package gère les dépendances

▷ La commande principale de l'advanced packaging tool est **apt-get**

  ▷ **apt-get install foo** permet d'installer l'application **foo**

  ▷ **apt-get source foo** permet de récupérer les sources de l'application **foo**

SOURCES.LIST

- La commande **apt-get** se base sur le fichier **/etc/apt/sources.list**
- Structure d'une ligne apt réelle

deb http://fr.archive.ubuntu.com/ubuntu/ trusty main restricted universe multiverse

- La premier bloc désigne le type de paquets contenus dans ce dépôt
- Deux mentions possibles:
  - **deb** qui désigne des paquets d'installation
  - **deb-src** qui désigne des paquets sources

SOURCES.LIST

- La commande **apt-get** se base sur le fichier **/etc/apt/sources.list**
- Structure d'une ligne apt réelle

deb http://fr.archive.ubuntu.com/ubuntu/ trusty main restricted universe multiverse

- Le second bloc désigne l'URL vers le serveur de paquets

SOURCES.LIST

- La commande **apt-get** se base sur le fichier **/etc/apt/sources.list**
- Structure d'une ligne apt réelle

deb http://fr.archive.ubuntu.com/ubuntu/ trusty main restricted universe multiverse

- Le troisième bloc identifie votre version d'Ubuntu

SOURCES.LIST

▷ La commande **apt-get** se base sur le fichier **/etc/apt/sources.list**

▷ Structure d'une ligne apt réelle

deb http://fr.archive.ubuntu.com/ubuntu/ trusty main restricted universe multiverse

▷ Le dernier bloc identifie les sections du dépôt auxquelles vous souhaitez accéder

▷ Les paquets sont divisés en section selon qu'ils sont maintenus par l'équipe d'Ubuntu ou par la communauté des utilisateurs d'Ubuntu, ainsi que selon leur licence

Par l'équipe d'Ubuntu et libres : **main**
Par l'équipe d'Ubuntu et non libres : **restricted**
Par les utilisateurs d'Ubuntu et libres : **universe**
Par les utilisateurs d'Ubuntu et non libres : **multiverse**

MISE À JOUR

▷ Une fois une mise à jour du fichier **sources.list** effectuée, il faut que **apt-get** mette à jour ses informations locales via la commande **apt-get update**

▷ La mise à jour d'applications se fait par la commande

apt-get upgrade

▷ Pour créer un paquet Debian, il faut suivre une recette bien particulière : Packaging.mov (8min)

GESTION DES DÉPENDANCES

▷ Il existe deux types de dépendances

▷ A la compilation (e.g., avec les fichiers d'en-têtes)

```
$> objdump -p /usr/bin/xeyes | grep NEEDED
NEEDED               libXext.so.6
NEEDED               libXmu.so.6
NEEDED               libXt.so.6
NEEDED               libX11.so.6
NEEDED               libXrender.so.1
NEEDED               libm.so.6
NEEDED               libc.so.6
```

▷ A l'exécution (e.g., besoin d'un autre programme pour s'exécuter) décrit dans la documentation

▷ Tout ceci est défini dans le fichier **debian/control**

GESTION DES DÉPENDANCES

▷ Pour chaque dépendance, on explicitera une dépendance vers les paquets **-dev** correspondant (e.g., **libxml2-dev**)

▷ Les dépendances implicites (liés au C) sont définies dans le paquet **build-essential**

▷ Lors de l'appel à **apt-get source foo**, on ne récupère que les sources de **foo**

▷ Pour obtenir les dépendances à la compilation, il faut effectuer la commande **apt-get build-dep foo**

GESTION DES DÉPENDANCES - EXEMPLE

▷ checkaz

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char**argv){
  char c;
  char command[32];
  for(c='a';c<='z';c++){
    sprintf(command,"checklib%c",c);
    system(command);
  }
  return EXIT_SUCCESS;
}
```

▷ Dépendance à l'exécution mais pas à la compilation

GESTION DES DÉPENDANCES - EXEMPLE

▷ Edition du fichier **debian/control**

```
Source: checkaz
Section: misc
Priority: optional
Maintainer: Guillaume Blin <gblin@emi.u-bordeaux1.fr>
Build-Depends: debhelper (>= 9)
Standards-Version: 3.9.6
Homepage: http://guillaume.blin.emi.u-bordeaux.fr

Package: checkaz
Architecture: all
Depends: ${shlibs:Depends}, ${misc:Depends}, checklib (>=1.0)
Description: simple so loadability testing
 Program that tests the loadability of .so files named with
 a single letter
```

TEST DU PAQUET (EN MODE ROOT)

```
$>apt-get install checkaz
Reading package lists... Done
Building dependency tree... Done
The following extra packages will be installed:
  checklib
The following NEW packages will be installed:
  checkaz checklib
0 upgraded, 2 newly installed, 0 to remove and 6 not upgraded.
Need to get 0B/5508B of archives.
After unpacking 81.9kB of additional disk space will be used.
Do you want to continue [Y/n]? Y
WARNING: The following packages cannot be authenticated!
  checklib checkaz
Install these packages without verification [y/N]? y
Selecting previously deselected package checklib.
(Reading database ... 54055 files and directories currently installed.
Unpacking checklib (from.../checklib_1.0-1_all.deb) ...
Selecting previously deselected package checkaz.
Unpacking checkaz (from.../archives/checkaz_1.0-1_all.deb) ...
Setting up checklib (1.0-1) ...
Setting up checkaz (1.0-1)...
```

SUPPRESSION DU PAQUET

▷ Si on supprime **checkaz**, le paquet **checklib** restera

▷ Si on supprime **checklib**, **apt-get** supprimera tout ce qui dépend de **checklib**

```
$>apt-get remove checklib
Reading package lists... Done
Building dependency tree... Done
The following extra packages will be REMOVED:
  checkaz checklib
0 upgraded, 0 newly installed, 2 to remove and 6 not upgraded.
Need to get 0B of archives.
After unpacking 81.9kB disk space will be freed.
Do you want to continue [Y/n]? Y
(Reading database... 54063 files and directories currently installed.)
Removing checkaz ...
Removing checklib ...
```

DOGGY BAG

TO TAKE AWAY ...

▷ L'option **-l** de **gcc** permet de spécifier le nom du fichier bibliothèque à rechercher et charger

▷ Les bibliothèques statiques sont copiées dans l'exécutable à la compilation

▷ Les bibliothèques dynamiques sont chargées en mémoire à la demande durant l'exécution du programme

Questions?