Cours de C - IR1
2007-2008

Bibliothèques
&
packaging

Sébastien Paumier

MCours.com

Les bibliothèques

Qu'est-ce que c'est ?

- bibliothèque=boîte noire capable de rendre des services
- 2 aspects:
  - le code (fichier binaire)
  - la liste des fonctions, types, variables (beurk) et constantes utilisables (fichier `.h`)
- fichier `libbiniou.a` ou `.so`

Utiliser une bibliothèque

1) inclure les `.h` nécessaires

2) indiquer au compilateur qu'il doit utiliser la bibliothèque **`biniou`** (**`libbiniou.a`** ou **`libbiniou.so`**):

```
gcc ..... -lbiniou
```

3) si le fichier n'est pas dans **`/usr/lib`**, il faut indiquer son chemin avec **`-Lchemin`**

Bibliothèque statique

- bibliothèque statique=fichier `.a` contenant un ou plusieurs fichier(s) `.o`
- à la compilation, les portions de code nécessaires sont copiées dans l'exécutable

\+ l'exécutable n'a plus besoin de la bibliothèque

– redondance de code entre les exécutables (d'où MAJ sans effet)

- peu utilisé, intérêt surtout historique

Exemple

- **utf8.c** / **utf8.h**: lecture et 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.
 *
 * NOTE: as it takes a unichar, this function cannot be used for writing
 *       a unicode character > 0xFFFF */
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.
 *
 * IMPORTANT: This function allows reading characters > 65536, so if
 *            it is used only for 16 bits unicode, the caller
 *            must check that the value is not greater than expected. */
int fgetc_utf8(FILE* stream);
```

Exemple

1) créer le fichier `.o`:

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

2) créer la bibliothèque `.a`:

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

3) visualiser son contenu:

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

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

Test

serait inutile si **libutf8.a** était dans **/usr/lib**

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

test.c

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

int main(int argc,char* argv[]) {
   /* é (code hexa E9) doit s'afficher
    * Ã© si le terminal n'est pas en
    * UTF8 */
   fputc_utf8(0xE9,stdout);
   fputc_utf8('\n',stdout);
   return 0;
}
```

Bibliothèque partagée

- fichier `.so` (**s**hared **o**bject)
- à l'exécution l'éditeur de liens dynamique ira chercher le code dans le `.so`
- économie: si plusieurs programmes partagent le code, il n'est qu'une fois en mémoire
- en cas de MAJ de la bibliothèque, les exécutables en profitent automatiquement

Exemple

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

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

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

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

3) création de l'exécutable:

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

Exemple

4) exécution qui ne marche pas:

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

5) explication avec **ldd** (affichage des 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)
```

pas dans **/usr/lib**, donc l'éditeur dynamique ne sait pas où chercher

Solution

- 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 (0xb7e8d000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0xb7fea000)
$>./a.out
Ã©
```

Nommage

- *linker name*: `libutf8.so`
  - nom de fichier utilisé pour compiler un exécutable
- *soname*: linker name+numéro de version majeur: `libutf8.so.1`
  - même numéro de version=update possible
  - si on remplace `libutf8.so.1` par `libutf8.so.2`, certains programmes risquent de ne plus fonctionner

Nommage

- *real name*: soname+numéro de version mineur+numéro optionnel de release: `libutf8.so.1.0.2` ou `libutf8-1.0.2.so`
- c'est ici que se trouve vraiment le code
- les autres noms sont des liens symboliques

```
$>ls -l /usr/lib/libm.so
lrwxrwxrwx  1 root root 19 jui 23  2004 /usr/lib/libm.so -> ../../lib/libm.so.6
$>ls -l /lib/libm.so.6
lrwxrwxrwx  1 root root 13 jui 23  2004 /lib/libm.so.6 -> libm-2.3.3.so
$>ls -l /lib/libm-2.3.3.so
-rwxr-xr-x  1 root root 141380 fév 16  2004 /lib/libm-2.3.3.so
```

Bibliothèque dynamique

- DL=Dynamically Loaded Libraries
- fichier `.so` chargé à l'exécution du programme
- permet la mise en place de plugins
- pratique pour écrire un JIT (Just In Time compiler):
  - compiler le code
  - charger le fichier objet
  - exécuter le code

Bibliothèque dynamique

- fonctions définies dans **`dlfcn.h`**
- compiler avec **`-ldl`**
- **`void* dlopen(const char* filename,int flag);`**
- charge la DL indiquée et retourne un pointeur la désignant, ou **`NULL`** si erreur
- **`flag`**: **`RTLD_LAZY`**=résolution quand nécessaire, **`RTLD_NOW`**=résolution de tous les noms de symboles utilisés dans la DL

Bibliothèque dynamique

- `int dlclose(void* handle);`
- décharge la DL indiquée si elle n'est plus utilisée par aucun programme
- `void* dlsym(void* handle,char* symbol);`
- cherche le symbole indiqué et le retourne, ou **`NULL`** si non trouvé

Bibliothèque dynamique

- `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 **`dlerror`**

Exemple

- localisation de **hello_world**:

hello_world_fr.c

```
#include <stdio.h>

void hello_world() {
   printf("Bonjour monde!\n");
}
```

hello_world_en.c

```
#include <stdio.h>

void hello_world() {
   printf("Hello world!\n");
}
```

- préparation des DL:

```
$>gcc -fPIC -c hello_world_fr.c
$>gcc -shared -o libhello_world_fr.so hello_world_fr.o
$>gcc -fPIC -c hello_world_en.c
$>gcc -shared -o libhello_world_en.so hello_world_en.o
```

Exemple

test.c

il faudrait tester tous les retours de fonction

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

int main(int argc,char* argv[]) {
void* dl;
void (*hello)(void);
dl=dlopen("./libhello_world_fr.so",RTLD_LAZY);
hello=dlsym(dl,"hello_world");
hello();
dlclose(dl);
dl=dlopen("./libhello_world_en.so",RTLD_LAZY);
hello=dlsym(dl,"hello_world");
hello();
dlclose(dl);
dl=dlopen("./libhello_world_de.so",RTLD_LAZY);
fprintf(stderr,"%s\n",dlerror());
return 0;
}
```

```
$>gcc test.c -ldl
$>./a.out
Bonjour monde!
Hello world!
./libhello_world_de.so: cannot open shared object file: No such file or directory
```

Bibliothèques et visibilité

- même si un élément n'est pas déclaré dans un `.h`, il est accessible
- exemple: `hello_world` n'était pas déclaré
- pour rendre un élément non visible, il faut le déclarer avec `static`
- sert à interdire l'accès à l'implémentation

Exemple

hello_world_fr.c

```
#include <stdio.h>

void hello_world() {
printf("Bonjour monde!\n");
}

static void hello_world2() {
printf("Bonjour monde2!\n");
}
```

test.c

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

int main(int argc,char* argv[]) {
void* dl;
void (*hello)(void);
dl=dlopen("./libhello_world_fr.so",
          RTLD_LAZY);
hello=dlsym(dl,"hello_world");
hello();
hello=dlsym(dl,"hello_world2");
fprintf(stderr,"%s\n",dlerror());
fclose(dl);
return 0;
}
```

```
$>gcc test.c -ldl
$>./a.out
Bonjour monde!
./a.out: undefined symbol: hello_world2
```

Les bibliothèques sous Windows

(tests effectués avec les outils GPL de Dev-C++ 4.9.8.0, téléchargeable à: **http://igm.univ-mlv.fr/~unitex/devcpp4980.exe**)

Bibliothèque statique

- mêmes opérations que sous Linux:

```
D:\lib_win>gcc -c utf8.c

D:\lib_win>ar rs libutf8.a utf8.o

D:\lib_win>gcc test.c -L. -lutf8

D:\lib_win>a.exe
├®
```

Bibliothèque partagée

- pas de `.so`, mais même idée que les DL
- DLL: Dynamic Link Library

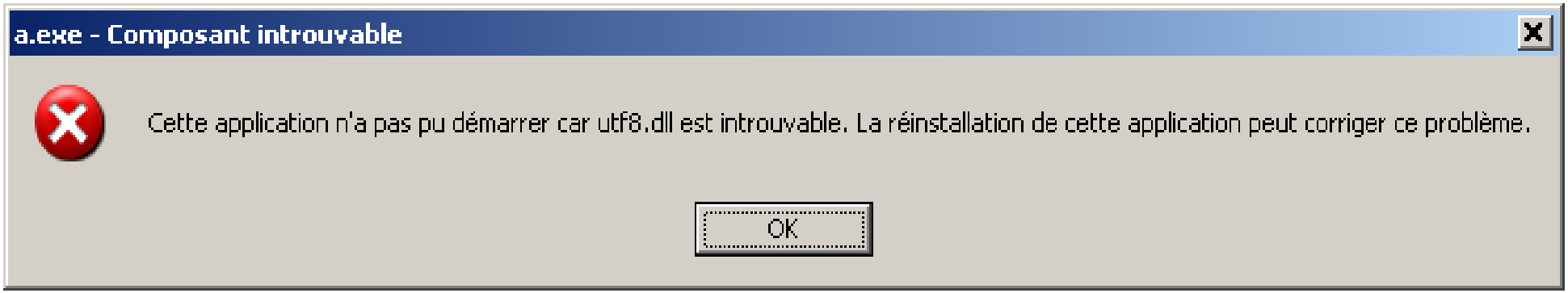
```
D:\lib_win>gcc -fPIC -c utf8.c
cc1.exe: warning: -fPIC ignored for target (all code is position
independent)

D:\lib_win>gcc -shared utf8.o -Wl,--export-all-symbols -o utf8.dll

D:\lib_win>gcc test.c -L. -lutf8

D:\lib_win>a.exe
├®
```

on pourrait restreindre en ne déclarant que les symboles souhaités

pas de préfixe `lib`

Bibliothèque partagée

- à l'exécution, le programme doit trouver la dll:
  - soit dans le même répertoire que lui
  - soit dans un des chemins indiqués par la variable d'environnement **PATH**
- sinon:



Bibliothèque dynamique

- même principe que sous Linux

```
#include <stdio.h>
#include <windows.h>

typedef void (*imported_type)(void);

int main(int argc,char* argv[]) {
   imported_type hello;
   HINSTANCE lib=LoadLibrary("hello_world_fr.dll");
   hello=(imported_type)GetProcAddress(lib,"hello_world");
   hello();
   FreeLibrary(lib);
   lib=LoadLibrary("hello_world_en.dll");
   hello=(imported_type)GetProcAddress(lib,"hello_world");
   hello();
   FreeLibrary(lib);
   return 0;
}
```

include différent

cast nécessaire

Bibliothèque dynamique

- même règle d'accessibilité des dll que précédemment

```
D:\lib_win>gcc -c hello_world_fr.c hello_world_en.c

D:\lib_win>gcc -shared hello_world_fr.o -Wl,--export-all-symbols
-o hello_world_fr.dll

D:\lib_win>gcc -shared hello_world_en.o -Wl,--export-all-symbols
-o hello_world_en.dll

D:\lib_win>gcc hello.c

D:\lib_win>a.exe
Bonjour monde!
Hello world!
```

Les packages selon Debian/Ubuntu

Principe

- les applications/bibliothèques sont packagées sous forme binaire et/ou source sur des *repositories*
- gestion des dépendances
- commande miracle: **apt-get** (**a**dvanced **p**ackaging **t**ool)

```
apt-get install pouet
apt-get source pouet
```

La commande miracle

- **apt-get**: à utiliser en mode root
- se base sur **/etc/apt/sources.list**:

binaires

URI

composants

```
deb ftp://debian.univ-mlv.fr/debian stable main contrib non-free

deb-src http://archive.debian.org/debian-archive hamm main
```

sources

distribution

Mises à jour

- MAJ de `sources.list`: il faut qu'`apt-get` mette à jour ses infos locales

```
apt-get update
```

- MAJ d'une application: si on veut tester la présence de MAJ des paquets déjà installés

```
apt-get upgrade
```

Créer son propre paquet

- exemple: **checklib**

checklib.c

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

int main(int argc,char* argv[]) {
   void* dl;
   char lib[1024];
   int i;
   for (i=1;i<argc;i++) {
      sprintf(lib,"lib%s.so",argv[i]);
      dl=dlopen(lib,RTLD_LAZY);
      printf("%s: %s\n",lib,(dl!=NULL)?"OK":"not found");
      if (dl!=NULL) {
         dlclose(dl);
      }
   }
   return 0;
}
```

Créer son propre paquet

Makefile

```
DESTDIR=
BINDIR=/usr/bin

all: checklib

checklib: checklib.c
	gcc checklib.c -o checklib -ldl

clean:
	rm -f checklib

install:
	install -d -m 0755 -o root -g root $(DESTDIR)/$(BINDIR)
	install    -m 0755 -o root -g root checklib $(DESTDIR)/$(BINDIR)
```

le répertoire d'installation doit être **DESTDIR**

Créer son propre paquet

- étape 1: créer un tar.gz contenant les sources et le Makefile

```
$>ls checklib-1.0/
Makefile  checklib.c
$>tgz checklib_1.0.orig.tar.gz checklib-1.0/
Writing gzip'd tar archive to "checklib_1.0.orig.tar.gz".
checklib-1.0/
checklib-1.0/Makefile
checklib-1.0/checklib.c
 95.0%
Total bytes written: 10240 (10KiB, 1,2MiB/s)
-rw-r--r-- 1 paumier ens 526 2007-11-07 15:49 checklib_1.0.orig.tar.gz
```

la forme du nom est importante

Créer son propre paquet

- étape 2: créer le squelette du paquet avec **dh_make**

```
$>cd checklib-1.0/
$>dh_make
ldap_sasl_interactive_bind_s: No such attribute (16)

Type of package: single binary, multiple binary, library, kernel
module or cdbs?
 [s/m/l/k/b] s

Maintainer name  : PAUMIER Sebastien
Email-Address    : paumier@kapouer.univ-mlv.fr
Date             : Wed,  7 Nov 2007 15:51:18 +0100
Package Name     : checklib
Version          : 1.0
License          : blank
Type of Package  : Single
Hit <enter> to confirm:
```

Créer son propre paquet

- après confirmation, le squelette est prêt à être édité

(suite de l'écran précédent)

```
Skipping creating ../checklib_1.0.orig.tar.gz because it already exists
Done. Please edit the files in the debian/ subdirectory now. You should also
check that the checklib Makefiles install into $DESTDIR and not in / .
```

ceci explique la remarque sur le Makefile

```
$>ls -l
total 12
-rw-r--r-- 1 paumier ens  341 2007-11-07 15:40 checklib.c
drwxr-xr-x 2 paumier ens 4096 2007-11-07 15:51 debian
-rw-r--r-- 1 paumier ens  251 2007-11-07 15:43 Makefile
```

Créer son propre paquet

```
$>ls debian
changelog              emacsen-remove.ex   postrm.ex          utf8-default.ex
compat                 emacsen-startup.ex  preinst.ex         utf8-dev.dirs
control                init.d.ex           prerm.ex           utf8-dev.install
copyright              manpage.1.ex        README.Debian      utf8.doc-base.EX
cron.d.ex              manpage.sgml.ex     rules              watch.ex
dirs                   manpage.xml.ex      shlibs.local.ex
docs                   menu.ex             utf81.dirs
emacsen-install.ex     postinst.ex         utf81.install
```

- étape 3: nettoyer le répertoire **debian** en supprimant les **.ex** et **.EX** (fichiers d'exemple), **dirs**, **docs** et **README.Debian**

Créer son propre paquet

- étape 4: édition de **control**

```
Source: checklib
Section: misc
Priority: optional
Maintainer: PAUMIER Sebastien <paumier@kapouer.univ-mlv.fr>
Build-Depends: debhelper (>= 5)
Standards-Version: 3.7.2

Package: checklib
Architecture: all
Depends:
Description: program that tests the loadability of .so files
```

Créer son propre paquet

- étape 5: édition de **changelog**

```
checklib (1.0-1) unstable; urgency=low

  * Initial release.

 -- PAUMIER Sebastien <paumier@kapouer.univ-mlv.fr>  Wed,  7 Nov
2007 14:12:07 +0100
```

- étape 6: édition de **copyright**

```
This package was debianized by PAUMIER Sebastien
<paumier@kapouer.univ-mlv.fr> on Wed,  7 Nov 2007 15:51:18 +0100.

Upstream Author: PAUMIER Sebastien <paumier@kapouer.univ-mlv.fr>
Copyright: 2007 Sebastien PAUMIER
License: public domain
The Debian packaging is (C) 2007, PAUMIER Sebastien
<paumier@kapouer.univ-mlv.fr> and
is licensed under the GPL, see `/usr/share/common-licenses/GPL'.
```

Créer son propre paquet

- étape 7: construire le paquet
  - se placer dans **`checklib-1.0`**
  - exécuter **`dpkg-buildpackage -rfakeroot`** (option à utiliser si l'on n'est pas root)
- résultat: plein de fichiers créés

voilà notre paquet!

```
$>ls .. -l
total 32
drwxr-xr-x 3 paumier ens 4096 2007-11-07 16:06 checklib-1.0
-rw-r--r-- 1 paumier ens 2856 2007-11-07 16:06 checklib_1.0-1_all.deb
-rw-r--r-- 1 paumier ens 1392 2007-11-07 16:06 checklib_1.0-1.diff.gz
-rw-r--r-- 1 paumier ens  329 2007-11-07 16:06 checklib_1.0-1.dsc
-rw-r--r-- 1 paumier ens  733 2007-11-07 16:06 checklib_1.0-1_i386.changes
-rw-r--r-- 1 paumier ens  526 2007-11-07 15:49 checklib_1.0.orig.tar.gz
```

Créer son repository

- arborescence spéciale, mais emplacement libre

```
$>mkdir -p ~/WWW/tutu/dists/unstable/main/binary-i386
$>mkdir -p ~/WWW/tutu/dists/unstable/main/source
```

- copie des fichiers générés précédemment

```
$>cp checklib_1.0-1.dsc ~/WWW/tutu/dists/unstable/main/binary-i386
$>cp checklib_1.0-1_all.deb ~/WWW/tutu/dists/unstable/main/binary-i386
$>cp checklib_1.0-1.diff.gz ~/WWW/tutu/dists/unstable/main/source
$>cp checklib_1.0-1.dsc ~/WWW/tutu/dists/unstable/main/source
$>cp checklib_1.0.orig.tar.gz ~/WWW/tutu/dists/unstable/main/source
```

Créer son repository

- génération de **Packages.gz** et **Sources.gz**

```
$>cd ~/WWW/tutu/dists/unstable/main
$>dpkg-scanpackages binary-i386 /dev/null dists/unstable/main/ | gzip -f9
> binary-i386/Packages.gz
$>dpkg-scansources source /dev/null dists/unstable/main/ | gzip -f9 >
source/Sources.gz
```

- création des fichiers de description du dépôt:

  binary-i386/Release

  source/Release

Créer son repository

- **binary-i386/Release**:

```
Archive : unstable
Version : 1
Component : main
Origin : Test
Label : checklib
Architecture : i386
```

- **source/Release**:

```
Archive : unstable
Version : 1
Component : main
Origin : Test
Label : checklib
Architecture : source
```

Utiliser son repository

- si on est root:
- mettre à jour **`/etc/apt/sources.list`** en ajoutant ces 2 lignes:

```
deb http://etudiant.univ-mlv.fr/~paumier/tutu unstable main
deb-src http://etudiant.univ-mlv.fr/~paumier/tutu unstable main
```

- et en faisant **`apt-get update`**
- il ne reste plus qu'à faire
  - soit **`apt-get install checklib`**
  - soit **`apt-get source checklib`**

Utiliser son repository

- si on n'est pas root:
- il faut simuler un environnement pour **apt-get**:

```
$>mkdir my_apt
$>cd my_apt
$>mkdir -p state/lists/partial
$>mkdir -p cache/archives/partial
```

- on crée ensuite **my_apt/toto.list** qui contient les lignes **deb** et **deb-src** précédentes

Utiliser son repository

- on utilise ensuite **apt-get** avec nos propres paramètres:

```
$>apt-get --assume-yes -o Dir::Etc::sourcelist=./toto.list
                       -o Dir::State=./state
                       -o Dir::Cache=./cache
                       -o Debug::NoLocking=true
                       update
```

- n'étant pas root, on est obligé de récupérer les sources

```
$>mkdir ~/tmp
$>cd ~/tmp
$>apt-get --assume-yes -o Dir::Etc::sourcelist=../my_apt/toto.list
                       -o Dir::State=../my_apt/state
                       -o Dir::Cache=../my_apt/cache
                       -o Debug::NoLocking=true
                       source checklib
```

Utiliser son repository

- on récupère ainsi les choses suivantes:

```
$>ls -l
total 16
drwxr-xr-x 3 paumier ens 4096 2007-11-08 10:24 checklib-1.0
-rw-r--r-- 1 paumier ens 1392 2007-11-08 09:42 checklib_1.0-1.diff.gz
-rw-r--r-- 1 paumier ens  329 2007-11-08 09:42 checklib_1.0-1.dsc
-rw-r--r-- 1 paumier ens  526 2007-11-08 09:42 checklib_1.0.orig.tar.gz
```

- on peut alors compiler les sources et utiliser l'application:

```
$>cd checklib-1.0
$>make
gcc checklib.c -o checklib -ldl
$>./checklib m n
libm.so: OK
libn.so: not found
```

Gestion des dépendances

- 2 types de dépendances:
  - à la compilation (exemple: paquets de développement avec les `.h`)
  - à l'exécution (exemple: besoin d'un autre programme pour tourner)
- on les définit dans le fichier `debian/control`

Dépendances à la compilation

- comment les trouver?
- compiler le programme
- faire **`objdump -p` *`nom`* `| grep NEEDED`**
- exemple:

```
$>objdump -p /usr/X11R6/bin/xeyes |grep NEEDED
  NEEDED      libXmu.so.6
  NEEDED      libXt.so.6
  NEEDED      libSM.so.6
  NEEDED      libICE.so.6
  NEEDED      libXext.so.6
  NEEDED      libX11.so.6
  NEEDED      libm.so.6
  NEEDED      libc.so.6
```

Dépendances à la compilation

- on prend ensuite, s'il existent, les paquets **`-dev`** correspondants
- exemple: si on trouve une dépendance vers **`libxml2`**, on va mettre une dépendance vers **`libxml2-dev`**
- dépendances implicites vers le contenu du paquet **`build-essential`** (**`gcc`**, **`make`**, etc.)
- pour connaître la liste des paquets standards: **`http://packages.debian.org`**

Dépendances à la compilation

- lorsqu'on fait **`apt-get source biniou`**, on ne récupère que les sources de **`biniou`**
- pour obtenir les dépendances à la compilation, il faut faire:

```
apt-get build-dep biniou
```

ce qui installe les paquets requis pour compiler **`biniou`**

Dépendances à l'exécution

- comment les trouver?
- en épluchant le README de l'application, si elle n'est pas de vous

Exemple

- exemple: **checkaz**
- pas besoin de **checklib** à la compilation, mais à l'exécution

```
#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 0;
}
```

Exemple

- édition de **debian/control**:

```
Source: checkaz
Section: misc
Priority: optional
Maintainer: PAUMIER Sebastien <paumier@kapouer.univ-mlv.fr>
Build-Depends: debhelper (>= 5)
Standards-Version: 3.7.2

Package: checkaz
Architecture: all
Depends: checklib (>=1.0)
Description: test
```

relations possibles sur les numéros de version:

<<	inférieur
<=	inférieur ou égal
=	égal
>>	supérieur
>=	supérieur ou égal

Test du paquet (en mode root)

```
$>apt-get update
(...blablabla...)
$>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) ...
```

gestion des dépendances OK

Suppression du paquet

- si on supprime **checkaz**, **checklib** restera
- si on supprime **checklib**, **apt-get** supprime tout ce qui en dépend

```
$>apt-get remove checklib
Reading package lists... Done
Building dependency tree... Done
The following 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 ...
```

apt-get remove libc6 ?

```
$>apt-get remove libc6
Reading package lists... Done
Building dependency tree... Done
The following packages will be REMOVED:
  acpid adduser alacarte alsa-base alsa-utils apt apt-utils aptitude at
(...très long blabla...)
WARNING: The following essential packages will be removed.
This should NOT be done unless you know exactly what you are doing!
  apt libc6 (due to apt) libgcc1 (due to apt) libstdc++6 (due to apt)
(...long blabla...)
0 upgraded, 0 newly installed, 657 to remove and 0 not upgraded.
Need to get 0B of archives.
After unpacking 996MB disk space will be freed.
You are about to do something potentially harmful.
To continue type in the phrase 'Yes, do as I say!'
 ?] n
Abort.
```

très bonne pratique de développement pour éviter les accidents...

Pour plus d'infos

- Guide du nouveau responsable debian

  http://igm.univ-mlv.fr/~paumier/maint-guide.fr.pdf

- Le coin du développeur Debian

  http://www.debian.org/devel/

- La charte Debian

  http://www.debian.org/doc/debian-policy/