Programmation C

Bibliothèques et packaging

ESIPE Info 1ère année

Université de Marne-la-Vallée

Bibliothèque

- Bibliothèque = boite noire capable de rendre des services
- 2 aspects :

  - le code (fichier binaire)

  - la liste des fonctions, types, variables (horrible!) et constantes utilisables (fichier `foo.h`)
- fichier `foo.a` ou `foo.so`

Utiliser une bibliothèque

- inclure les `.h` si nécessaire
- indiquer au compilateur (linker) qu'il doit utiliser la bibliothèque `foo`
  (`foo.a` ou `foo.so`) :
  `gcc ..... -lfoo`
- 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 fichiers `.o`
- à la compilation, les portions de code nécessaires sont copiées dans l'exécutable

  - **avantage :** l'exécutable n'a plus besoin de la bibliothèque

  - **inconvénient :** redondance de code entre les exécutables (les mises à jour sont sans effet)
- peu utilisées, intérêt surtout historique

Exemple : Faire des piles pour un dc

Un module pour manipuler une pile d'entier

fichier **stack.h** :

```
 1  /* This module allows the manipulation of a single integer stack */
 2
 3  /* Initialize the stack at the empty stack */
 4  int stack_init(void);
 5
 6  /* Pop the top element of the stack and return its value */
 7  int stack_pop(void);
 8
 9  /* Push the given element at the top of the stack */
10  void stack_push(int n);
11
12  /* Return the value of the top element of the stack */
13  int stack_top(void);
14
15  /* Display all element contained in the stack */
16  void stack_display(void);
```

Exemple : une bibliothèque de pile

- créer le fichier objet `.o`

  ```
  $>gcc -c stack.c
  ```

- créer la bibliothèque `.a`

  ```
  $>ar rs libstack.a stack.o
  ```

- visualiser son contenu

```
nborie@perceval:~> nm --defined-only libstack.a

stack.o:
0000000000000000 b S
0000000000000106 T stack_display
0000000000000000 T stack_init
0000000000000010 T stack_pop
0000000000000063 T stack_push
00000000000000bf T stack_top
```

Test

```c
#include "stack.h"

int main(int argc, char* argv[]){
  stack_init();
  stack_push(1); stack_push(2); stack_push(3);
  stack_pop();
  stack_push(4); stack_push(5);
  stack_display();
  return 0;
}
```

```
nborie@perceval:~> gcc test.c
/tmp/ccKPCclN.o: dans la fonction main:
test.c:(.text+0x10): référence indéfinie vers stack_init
test.c:(.text+0x1a): référence indéfinie vers stack_push
 ....
nborie@perceval:~> gcc test.c -L. -lstack
nborie@perceval:~> ./a.out
1 2 4 5
```

L'option `-L.` serait inutile si `libstack.a` était bien dans `/usr/lib`

Bibliothèque partagée

- fichier `.so` (shared object)
- à l'exécution, l'éditeur de liens dynamiques ira chercher le code dans le `.so`
- économie : si plusieurs programmes partagent le code, il n'est qu'une seule fois en mémoire
- en cas de mise à jour de la bibliothèque, tous les exécutables en profitent automatiquement

Exemple

- créer le fichier `.o` avec l'option `fPIC` (`P`osition `I`ndependant `C`ode)

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

- créer la bibliothèque avec l'option `-shared` :

  ```
  $>gcc -shared -o libstack.so stack.o
  ```

- création de l'exécutable

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

Exemple

- exécution qui ne marche pas :

```
nborie@perceval:~> ./a.out
./a.out: error while loading shared libraries: libstack.so:
cannot open shared object file: No such file or directory
```

- explication avec `ldd` (affichage des dépendances):

```
nborie@perceval:~> ldd a.out
linux-vdso.so.1 =>  (0x00007fffddce0000)
libstack.so => not found
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f413b887000)
/lib64/ld-linux-x86-64.so.2 (0x00007f413bc6f000)
```

Le linker dynamique ne peut trouver la bibliothèque `libstack.so` car celle-ci ne se trouve pas dans `/usr/lib`

Solution

- si on n'a pas accès à `/usr/lib`, on doit compiler l'exécutable avec l'option `-Wl, -rpath, chemin_du_so` :

```
nborie@perceval:~> gcc test.c -L. -lstack -Wl,-rpath,.
nborie@perceval:~> ./a.out
1 2 4 5
nborie@perceval:~> ldd a.out
linux-vdso.so.1 =>  (0x00007fff4effe000)
libstack.so => ./libstack.so (0x00007ffd74aa8000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ffd746c2000)
/lib64/ld-linux-x86-64.so.2 (0x00007ffd74cac000)
```

Nommage

- **Linker name** : `libstack.so`

  - nom de fichier utilisé pour compiler un exécutable

- **Soname** : linker name + numéro de la version majeure : `libstack.so.1`

  - même numéro de version = update possible (les différentes versions d'une même majeure ne doivent pas casser la backward compatibility)

  - si on remplace `libstack.so.1` par `libstack.so.2`, certains programmes risquent de ne plus fonctionner.

Nommage

- **Real name** : soname + numéro de version mineure + numéro optionnel de release : `libstack.so.1.0.2` ou bien `libstack-1.0.2.so`
- C'est ici que se trouve vraiment le code !
- Les autres noms ne sont souvent que les liens symboliques

```
nborie@perceval:~/ > ls -l /usr/lib/libMLV.so
lrwxrwxrwx 1 root 15 juil. 12  2012 /usr/lib/libMLV.so -> libMLV.so.0.0.0
nborie@perceval:~/ > ls -l /usr/lib/libMLV.so.0
lrwxrwxrwx 1 root 15 juil. 12  2012 /usr/lib/libMLV.so.0 -> libMLV.so.0.0.0
nborie@perceval:~/ > ls -l /usr/lib/libMLV.so.0.0.0
-rw-r--r-- 1 root 209296 juil. 12  2012 /usr/lib/libMLV.so.0.0.0
```

Bibliothèque dynamique

- DL = Dynamically Loaded Libbrairies
- 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);`
- `dlopen` 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, const char *symbol);`
- cherche le symbole indiqué et le retourne, ou retourne `NULL` si non trouvé

Bibliothèque dynamique

- `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

```
1 #include <stdio.h>
2
3 void hello_world() {
4   printf("Bonjour nonde!\n");
5 }
```

hello_world_en.c

```
1 #include <stdio.h>
2
3 void hello_world() {
4   printf("Hello world!\n");
5 }
```

- préparation des DL :

```
nborie@perceval:~> gcc -fPIC -c hello_world_fr.c
nborie@perceval:~> gcc -fPIC -c hello_world_en.c
nborie@perceval:~> gcc -shared -o libhello_world_fr.so hello_world_fr.o
nborie@perceval:~> gcc -shared -o libhello_world_en.so hello_world_en.o
nborie@perceval:~> ls
hello_world_en.c  hello_world_en.o  hello_world_fr.c  hello_world_fr.o
libhello_world_en.so  libhello_world_fr.so
```

Exemple

```
 1  #include <stdio.h>
 2  #include <dlfcn.h>
 3
 4  int main(int argc, char* argv[]){
 5    void* dl;
 6    void (*hello)(void);
 7    dl=dlopen("./libhello_world_fr.so", RTLD_LAZY);
 8    hello=dlsym(dl,"hello_world");
 9    hello();
10    dlclose(dl);
11    dl=dlopen("./libhello_world_en.so", RTLD_LAZY);
12    hello=dlsym(dl,"hello_world");
13    hello();
14    dlclose(dl);
15    dl=dlopen("./libhello_world_de.so", RTLD_LAZY);
16    fprintf(stderr, "%s\n", dlerror());
17    return 0;
18  }
```

```
nborie@perceval:~> gcc test.c -ldl
nborie@perceval:~> ./a.out
Bonjour nonde!
Hello world!
./libhello_world_de.so: cannot open shared object: No such file or directory
```

Bibliothèques et visibilité

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

Quelques mots pour windows

- les bibliothèques statiques se comportent de la même manière que sous `Unix`
- les bibliothèques partagées ne s'appellent pas "shared object" mais DLL (pour Dynamic Link Library), elles sont aussi compilées avec l'option `shared` mais l'option `-fPIC` n'est plus utile.
- les DLL n'ont pas de préfixe `lib` comme sous `Unix`, elles s'appellent ainsi `foo.dll`
- à l'exécution, la DLL doit se trouver dans le même répertoire que le fichier à compiler ou bien dans un des chemins indiqués par la variable d'environnement `PATH`

Quelques mots pour windows

- l'inclusion est alors :

  `#include <windows.h>`
  (c'est à dire toute l'API windows...)

- l'ouverture de la DLL se fait avec `LoadLibrary`

- les symboles peuvent être recherchés avec `GetProcAddress`

- les fonctions récupérées doivent encore être bien castées à l'assignation.

- Mêmes règles d'accessibilité que pour les DLL précédemment

Debian package : dpkg

- les applications/bibliothèques sont packagées sous forme de binaire et/ou source sur des *repositories*
- gestion des dépendances
- commande miracle : `apt-get`
  (advanced packaging tool)

```
sudo apt-get install foo
sudo apt-get source foo
sudo apt-get remove foo
```

La commande miracle

- `apt-get` : à utiliser en mode root
- se base sur `/etc/apt/sources.list` :
- (binaire) (URI) (composants)

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

- (sources) (URI) (distribution)

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

Mises à jour

- Lors d'une MAJ de `sources.list` : `apt-get` doit mettre à jour sa copie locale de la liste des paquets disponibles

  ```
  sudo apt-get update
  ```

- Lors d'une MAJ dans les `repositories` d'une ou plusieurs applications déjà installées, on peut demander au système de tout mettre à jour

  ```
  sudo apt-get upgrade
  ```

Créer son propre paquet

Exemple (relativement super utile) : `minus_and_cortex`

`minus_and_cortex.c` :

```
1  #include <stdio.h>
2
3  /* This function displays a simple message on the
4     standard output. this constitutes the first step
5     of my plan to take over the world */
6  int main(int argc, char* argv[]){
7    printf("Hey Pinky, as Debian dev, tomorrow "
8           "evening, we will take over the world!\n");
9    return 0;
10 }
```

Créer son propre paquet

Makefile :

```
DESTDIR=
BINDIR=/usr/bin

all: minus_and_cortex

minus_and_cortex: minus_and_cortex.c
	gcc minus_and_cortex.c -o minus_and_cortex -ldl

clean:
	rm -f minus_and_cortex

install:
	install -d -m 0775 -o root -g root $(DESTIR)/$(BINDIR)
	install    -m 0775 -o root -g root minus_and_cortex $(DESTIR)/$(BINDIR)
```

Le répertoire d'installation DOIT ÊTRE `DESTDIR`

Créer son propre paquet

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

```
nborie@perceval:~> ls minuscortex-1.0/
minus_and_cortex.c   Makefile
nborie@perceval:~> tgz minuscortex_1.0.orig.tar.gz minuscortex-1.0/
Writing gzip'd tar archive to "minuscortex_1.0.orig.tar.gz".
minuscortex-1.0/
minuscortex-1.0/minus_and_cortex.c
minuscortex-1.0/Makefile
Nombre total d'octets écrits: 10240 (10KiB, 5,6MiB/s)
94.9%
-rw-rw-r-- 1 nborie nborie 545 nov.  27 22:32 minuscortex_1.0.orig.tar.gz
```

Attention, la forme du nom `minuscortex_1.0.orig.tar.gz` est importante.

Créer son propre paquet

- étape 2 : créer le squelette du paquet avec dh_make

```
nborie@perceval:~> ls
minus_and_cortex.c   Makefile
nborie@perceval:~> dh_make

Type of package: single binary, indep binary, multiple binary,
library, kernel module, kernel patch?
 [s/i/m/l/k/n] s

Maintainer name  : nborie
Email-Address    : nborie@unknown
Date             : Sun, 01 Dec 2013 21:30:09 +0100
Package Name     : minuscortex
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
- bien lire les messages d'information des commandes de construction de paquet
- `dh_make` a créé un répertoire `Debian` contenant des fichiers relatifs à la politique de développement des paquets.

```
Skipping creating ../minuscortex_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 minuscortex Makefiles install into $DESTDIR and not in / .
```

Créer son propre paquet

```
nborie@perceval:~/test_ppa2/minuscortex2-1.0/ > ls -l debian
total 100
changelog                       compat
control                         copyright
docs                            init.d.ex
manpage.1.ex                    manpage.sgml.ex
manpage.xml.ex                  menu.ex
minuscortex2.cron.d.ex          minuscortex2.default.ex
minuscortex2.doc-base.EX        postinst.ex
postrm.ex                       preinst.ex
prerm.ex                        README.Debian
README.source                   rules
source                          watch.ex
```

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

Créer son propre paquet

- étape 4: édition de `control`

```
nborie@perceval:~> cat control
Source: minuscortex
Section: unknown
Priority: extra
Maintainer: nborie <nborie@unknown>
Build-Depends: debhelper (>= 8.0.0)
Standards-Version: 3.9.4
Homepage: <insert the upstream URL, if relevant>
#Vcs-Git: git://git.debian.org/collab-maint/minuscortex.git

Package: minuscortex
Architecture: any
Depends:
Description: <insert up to 60 chars description>
 <insert long description, indented with spaces>
```

Créer son propre paquet

- étape 5 : édition de `changelog`

```
nborie@perceval:~> cat changelog
minuscortex (1.0-1) unstable; urgency=low

  * Initial release (Closes: #nnnn)  <nnnn is the bug number of your ITP>

 -- nborie <nborie@unknown>  Sun, 01 Dec 2013 21:30:09 +0100
```

- étape 6 : édition de `copyright`

```
nborie@perceval:~> cat copyright
Format: http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: minuscortex
Source: <url://example.com>

Files: *
Copyright: <years> <put author's name and email here>
           <years> <likewise for another author>
License: <special license>
 <Put the license of the package here indented by 1 space>
 <This follows the format of Description: lines in control file>
 .
 <Including paragraphs>
```

Créer son propre paquet

- étape 7 : construire le paquet

  - se placer dans `minuscortex-1.0/`

  - exécuter `dpkg-buildpackage -rfakeroot`
  (l'option est à utiliser si vous n'êtes pas root)

- Plein de fichiers créés

```
nborie@perceval:~/test_ppa2/ > ls -l
total 32
drwxrwxr-x  3 nborie nborie 4096 déc.   2 00:48 minuscortex-1.0
-rw-r--r--  1 root   root   1436 déc.   2 00:49 minuscortex_1.0-1_amd64.changes
-rw-r--r--  1 root   root   1148 déc.   2 00:49 minuscortex_1.0-1_amd64.deb
-rw-r--r--  1 root   root    905 déc.   2 00:48 minuscortex_1.0-1.debian.tar.gz
-rw-r--r--  1 root   root    780 déc.   2 00:48 minuscortex_1.0-1.dsc
-rw-rw-r--  1 nborie nborie 1320 déc.   2 00:48 minuscortex_1.0.orig.tar.gz
```

Créer son repository

- arborescence spéciale, mais emplacement libre

```
mkdir -p ~/WWW/tutu/dists/unstable/main/source
mkdir -p ~/WWW/tutu/dists/unstable/main/binary-amd64
```

- copie des fichiers générés précédemment

```
nborie@perceval:...> pwd
/var/www/pouet/dists/unstable/main
nborie@perceval:...> sudo cp ~/minuscortex_1.0-1_amd64.deb ./binary-amd64
nborie@perceval:...> sudo cp ~/minuscortex_1.0-1.dsc ./binary-amd64
nborie@perceval:...> sudo cp ~/minuscortex_1.0-1.dsc ./source
nborie@perceval:...> sudo cp ~/minuscortex_1.0.orig.tar.gz ./source
nborie@perceval:...> sudo cp ~/minuscortex_1.0-1_amd64.changes ./source
```

Créer son repository

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

```
nborie@perceval:...> pwd
/var/www/pouet/dists/unstable/main
nborie@perceval:...> sudo dpkg-scanpackages binary-amd64
/dev/null dists/unstable/main | gzip -f9 > binary-amd64/Packages.gz
avertissement: Paquets dans l'archive mais pas dans le fichier d'override :
avertissement:   minuscortex
info: 1 entrées écrites dans le fichier Packages.
nborie@perceval:...> sudo dpkg-scansources source
/dev/null dists/unstable/main | gzip -f9 > source/Sources.gz
```

- création des fichiers de description du dépôt :

  `binary-amd64/Release`

  `source/Release`

Créer son repository

- binary-amd64/Release

```
Archive : unstable
Version : 1
Component : main
Origin : Test
Label : minuscortex
Architecture : amd64
```

- source/Release

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

Utiliser son repository

- Si on est root:
- mettre à jour `/etc/apt/sources.list` en ajoutant les deux lignes

  ```
  deb http://localhost/pouet/ unstable main
  deb-src http://localhost/pouet/ unstable main
  ```

- on fait `sudo apt-get update`
- il ne reste plus qu'à faire :

  ```
  sudo apt-get install minuscortex
  sudo apt-get source minuscortex
  ```

Gestion des dépendances

- 2 types de dépendances:

  - à la compilation (exemple: paquets de développement avec des `.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 :

```
nborie@perceval:~> minus_and_cortex
Hey Pinky, as Debian dev, tomorrow evening, we will take over the world!
nborie@perceval:~> which minus_and_cortex
/usr/bin/minus_and_cortex
nborie@perceval:~> objdump -p /usr/bin/minus_and_cortex | grep NEEDED
  NEEDED               libc.so.6
```

Dépendances à la compilation

- on prend ensuite, s'ils existent, les paquets `-dev` correspondants
- exemple: si on trouve une dépendance vers `libc.so.6`, on va mettre une dépendance vers `libc6-dev`
- dépendances implicites vers le contenu du paquet `build-essential` (`gcc`, `make`, ...)
- Pour connaître la liste des paquets standards : voir http://www.debian.org/distrib/packages

Dépendances à la compilation

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

  `apt-get build-dep foo`

  ce qui installera les paquets requis pour compiler `foo`

Dépendance à l'exécution

- comment les trouver ?
- en épluchant la documentation de l'application si elle n'est pas de vous.
- le code source a toujours raison mais s'il faut aller jusqu'à le lire pour savoir, c'est que l'application est MAL documentée

Exemple

- Le formidable paquet : `invadenow`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char* argv[]){
5     char command[256];
6     sprintf(command, "minus_and_cortex");
7     system(command);
8     return 0;
9  }
```

Exemple

- édition de `debian/control` :

```
Source: invadenow
Section: misc
Priority: optional
Maintainer: nborie <nborie@dtc.com>
Build-Depends: debhelper (>= 8.0.0)
Standards-Version: 3.9.4

Package: invadenow
Architecture: any
Depends: minuscortex (>=1.0)
Description: minus and cortex attack now
```

relation sur les numéros de version possible

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

Suppression du paquet

- si on supprime `invadenow`, `minuscortex` restera
- si on supprime `minuscortex`, `apt-get` supprime tout ce qui en dépend
- la politique de développement veut, bien sûr, que tout paquet ne gérant pas parfaitement ces dépendances soit refusé (et heureusement!!!!)

Pour plus d'info

- le coin du développeur Debian
  - http://www.debian.org/devel/
- la charte Debian
  - http://www.debian.org/doc/debian-policy/