Compilation Séparée - Makefile

L. Mounier

UGA - M2 CCI - PL1

28 octobre 2021

Objectifs de la compilation séparée

Développement de programmes de “grande taille” . . .

- taille du code source ($>$ 500 loc ?)
- travail collaboratif, répartition géographique
- (ré)-utilisation de *composants logiciels*

$\Rightarrow$ Des conséquences pratiques

1. répartir le code source dans **plusieurs** fichiers
2. compiler (et tester) des **parties** du programmes

Principales motivations

- Taille du code source > taille d'un fichier, d'un répertoire ...
- Structure et lisibilité du code source
  un (ensemble de) fichier(s) source(s) par **composant**
  ex : affichage, calcul, sauvergarde, etc
- Travail collaboratif
  - édition du code source à plusieurs
  - indépendance des "composants" du programe
    partie publique (interface) / partie privée (implémentation)
  - test et mise au point de **parties** du logiciel
- Utilisation de code **externe**
  - ré-utilisation de composants **compilés**, **librairies**
  - confidentialité du code source, propriété intellectuelle
  - utilisation de plusieurs langages sources

Principe

Deux grandes étapes :

1. Compiler **indépendament** des parties du code source
   - un fichier source (f^s) $\rightarrow$ un fichier "compilé" (f^c)
   - les fichiers f^c :
     - contiennent des données et instruction en langage machine
     - sont **non exécutables** (car incomplets)
     - peuvent être "concaténés" pour produire des **bibiothèques**
2. "Fusionner" les parties compilées séparément
   - $(f_1^c, \ldots f_n^c) \rightarrow$ un fichier **exécutable** f
   - on parle d'**édition de liens** (*link*)

Mise en oeuvre en C

Fichiers d'en-tête (*headers*)

Ecrire dans un fichier `f.h` l'**interface** du fichier `f.c`
↪ définitions des types, variables et fonctions **publiques** du fichier

Compilation séparée

```
gcc -Wall -c f.c
```

→ produit le fichier compilé `f.o`

Edition de liens

```
gcc -Wall -o f f1.o f2.o ... fn.o
```

→ produit le fichier exécutable `f` à partir de `f1.o`, `f2.o`, ...`fn.o`

Remarque : compilation **simultanée** de # fichiers sources/compilés
`gcc -Wall -o f f1.c f2.o ... fn.c` (≠ compilation séparée !)

Un exemple simple

Trois fichiers `main.c`, `affiche.c` et `calcul.c`

- fichiers d'en-tête

```
/* fichier calcul.h */
int somme (int a, int b) ; // renvoie a+b
```

```
/* fichier affiche.h */
int affiche (int x) ; // affichage la valeur de x
```

- compilation séparée

```
gcc -Wall -c affiche.c
gcc -Wall -c calcul.c
gcc -Wall -c main.c
```

- édition de liens

```
gcc -Wall -o programme affiche.o calcul.o main.o
```

`make` : un outil d'aide à la compilation séparée

Objectifs

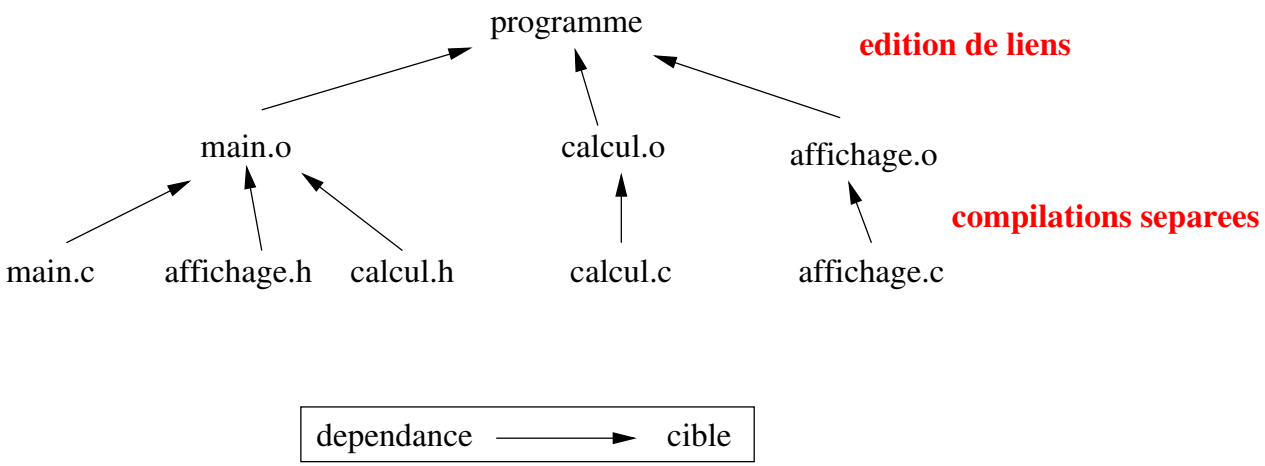
- grouper dans un seul fichier *Makefile* l'ensemble des commandes de compilation
- utiliser un outil (*make*) pour exécuter **uniquement** les commandes nécessaires à la (re)-compilation

Principes

- une **cible** et des **dépendances** associées à chaque commande
- si la cible est plus ancienne [1] qu'une des dépendances la commande est exécutée (sinon elle est ignorée)
- on spécifie la cible principale (par défaut la 1ere du `Makefile`)

1. date de création et/ou modification

Cibles et dépendances . . .



Structure du *Makefile* :

```
cible: liste de dépendances
 TAB commande pour produire la cible (à partir des dépendances)
```

Le *Makefile* complet

```
programme: main.o calcul.o affichage.o
    gcc -Wall -o programme main.o calcul.o affichage.o

main.o: main.c affichage.h addition.h
    gcc -Wall -c main.c

affichage.o: affichage.c
    gcc -Wall -c affichage.c

calcul.o: calcul.c
    gcc -Wall -c calcul.c
```

Attention aux tabulations en début de commande !

Exemple plus complet : les nombres complexes