

# Chaîne de compilation

## Compilateurs

Yves STADLER

Codasystem, UPV-M

13 septembre 2011

## Objectifs

- Maîtriser GCC
- Comprendre ce que fait GCC
- Différence entre compilation et édition de lien
- Maîtriser la compilation séparée
- Compilation dynamique vs. compilation statique

## Qu'est-ce que la chaîne de compilation

- En bref : ensemble des processus menant d'un code source à un code exécutable
- Traduction du langage vers les instructions

## Que fait le compilateur : GCC

Quatre opérations :

- Pré-processing (Preprocessing)
- Compilation (Compilation)
- Assemblage (Assembly)
- Édition de lien (Linking)

# Phase preprocessing

Que se passe-t-il : gcc -E

- Première passe
- Traite les fichiers d'en-têtes
- Traite les instruction de pré-compilation
- Expansion des macros
- Fichier .i

# Phase compilation

Que se passe-t-il : `gcc -S`

- Seconde passe
- Analyse lexicale (Génération des erreurs lexicales : instructions non reconnues,
- Analyse grammaticale
- Optimisations
- Génération code assembleur
- Fichier `.s`

# Que se passe-t-il : gcc -c

## Assemblage

- Listing d'assemblage avec offset (Suppression des symboles, remplacés par des adresses)
- Fichier .o

# Que se passe-t-il : gcc

## Édition de liens

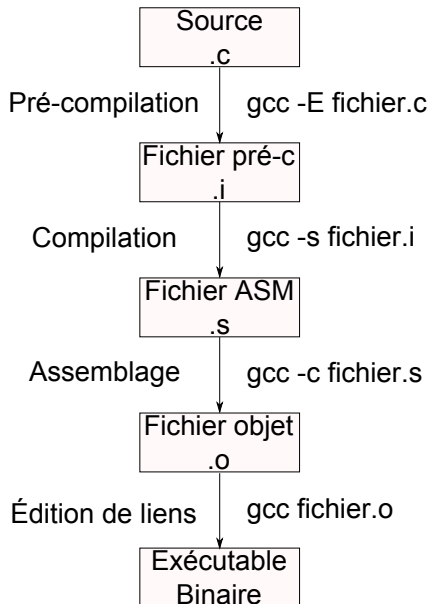
- Étape finale
- Un ou plusieurs fichiers objets/librairies
- Combine les codes pour créer un exécutable en résolvant les symboles externes.

## Note

- On peut utiliser gcc sur un fichier .o .s .i ou .c
- On peut utiliser gcc -c sur un fichier .s .i ou .c
- On peut utiliser gcc -s sur un fichier .i ou .c
- On peut utiliser gcc -E sur un fichier .c
- La compilation reprend à l'étape correspondante.



# En image



# Par la pratique

## En pratique

```
int main(void) {  
    return 0;  
}
```

`gcc -c ops.c`

- Créer le code ASM pour ce programme dans un fichier ops.o

`gcc`

- `gcc ops.o` donne l'exécutable

# Par la pratique

## Fichier supplémentaire

- On crée le fichier `lmath.c` contenant

```
int add(int a, int b) {  
    return a + b;  
}
```

```
int mul(int a, int b) {  
    return a * b;  
}
```

## obs.c

```
#include "lmath.c"  
int main(void) {  
    int res = add(1, 2);  
    res = add(2, 4);  
    return 0;  
}
```

# Par la pratique

```
gcc obs.c
```

- Reviens à mettre toutes les fonctions dans le même fichier
- Génère l'exécutable.

## Comment faire mieux ?

- On aimerai pouvoir compiler une partie de projet avec les fonctions add et mul et l'autre partie avec le main indépendamment.
- Principe de la compilation séparée

# Par la pratique

## Compilation séparée

- `gcc -c lmath.c`
- Génère un fichier `lmath.o` contenant le code des fonctions `add` et `mul`
- Comme on ne veut pas inclure le code des fonctions dans `obs.c`, il faut créer un fichier d'en-têtes `lmath.h`

```
int add(int a, int b);  
int mul(int a, int b);
```

- Ce fichier dit qu'il existe quelque part des fonctions nommées `add` et `mul`.

## obs.c

```
#include "lmath.h"  
int main(void) {  
    int res = add(1, 2);  
    res = mul(2, 4);  
    return 0;  
}
```

# Par la pratique

gcc obs.c

- Erreur !!
- Undefined reference to add...
- Undefined reference to mul...
- On a oublier de dire où se trouve le code assemblé de ces fonctions

gcc lmath.o obs.c

- Compile obs.c en obs.o
- Lie obs.o avec lmath.o et créer un exécutable.

# Par la pratique

## Alternative

- `gcc -c lmath.c`
- `gcc -c obs.c`
- `gcc obs.o lmath.o`

# Externe

## Variable externe

- Supposons que je souhaite avoir à ma disposition le dernier résultat calculé

## lmath.h

- Je définis la variable : `extern int lastResult;`
- Tout module qui importera lmath.h saura qu'il existe quelque part une variable nommée lastResult

## obs.c

```
#include "lmath.h"
int main(void) {
    add(1, 2);
    mul(lastRes, 4);
    return 0;
}
```



## Détails supplémentaires

- Lorsque l'on définit un prototype de fonction, il est externe implicitement !
- Lorsque l'on définit une variable, elle ne l'est pas

## Quand le répertoire prend l'o !

- Avec des grands projets, on se retrouve vite avec beaucoup de fichier .o
- Pourquoi on ne pourrait pas les regrouper dans un seul package pour les lier en une fois
- Et bien on peut grâce à ar
- Tous les fichiers archivés dans une librairie : libXXX.a (ar -ra libXXX.a fichier.o / ranlib libXXX.a)
- gcc fichier.c libXXX.a / gcc fichier.c -L. -lXXX
- Exemple la bibliothèque standard c libc.a

# Static sur la ligne !

## HelloWorld.c

```
#include <stdio.h>
int main(void) {
    printf("Hello World\n");
    return 0;
}
```

## Utilisation statique

- Le fichier header de stdio se trouve dans /usr/include
- Il indique que la fonction printf se trouve quelque part
- gcc -static HelloWorld.c
- Créer un programme standalone, qui contiendra le code de main, et celui de toutes fonctions appelées par printf.

# Static sur la ligne !

## C'est de la dynamique !

- On peut utiliser des bibliothèques partagées par plusieurs programmes
- Lors de l'exécution du programme, on fera la liaison avec ses fonctions (dynamique)

## Exemple

- `gcc -c -fPIC lmath.c`
- `gcc -shared -o liblmath.so lmath.o`
- `gcc -c main.c`
- `gcc main.o -L. -llmath`