Université de Strasbourg

S31 TD/TP N°6

Comprendre les Systèmes

[gancarski]@unistra.fr

IUT Robert Schuman Université de Strasbourg

Compilation séparée - Makefile

Objectif : Comprendre la compilation séparée et la compilation automatique

Notions : Makefile

---

1 Compilation séparée

Pour limiter la taille des fichiers source et permettre de compiler indépendamment les sous-parties d'un programme, on peut utiliser un mécanisme de compilation séparée : l'idée est de compiler indépendamment les fichiers sources qui contiennent chacun un ensemble "cohérent" de fonctions. Puis d'intégrer ce code compilé au programme final uniquement lors de la compilation de son code.
Soit une "gros" programme simulant une calculatrice (voir fichier `calc_init.c` sur Moodle).
Ce programme contient 5 includes, 3 fonctions et un main. On désire pouvoir le compiler par morceaux (compilation séparée). Il y a donc deux étapes :

1.1 - Créer des fichiers séparés pour les includes et les fonctions pour une compilation séparée :
- Créer les fichiers `calcule.c` et `calcule.h` pour permettre à une compilation séparée de la fonction `calcule`. Que contient le fichier `calcule.h` et à quoi sert-il ?
- Idem pour `SaisirOP` et `AfficherOP` dans les fichier `calculatrice_f_ext.h/c`
- Créer le fichier `myIncludes.h` qui regroupera tous les includes nécessaires au main.
- Créer un fichier `calculatrice.c` pour le `main`

1.2 - Compiler les fichiers et créer un exécutable `calculatrice`

2 Makefile

Il est possible "d'automatiser" la compilation. L'idée est de construire uniquement les fichiers de code en se limitant à ceux dont on en a besoin pour compiler le programme principal. En plus, on ne reconstruit que ceux qui ont été modifiés depuis leur dernière compilation. La commande `make` permet de définir des cible à construire (généralement par une compilation) et donner les conditions de leur construction. Construire un tel fichier pour la calculatrice.

2.1 Construire un fichier `makefile.txt` permettant de créer un exécutable `calculatrice` à partir des fichiers précédents.

3 Bibliothèque

3.1 Donner la commande permettant de construire la librairie statique `libcalc.a` intégrant des fichiers ".o" précédents.

3.2 Modifier votre fichier `makefile.txt` et recompiler le main en conséquence

Rappel : une librairie statique se construit par `libtool -static -o libcprog.a p1.o ...pn.o`

4 Makefile enrichi

Votre `makefile.txt` de base de peut être enrichi de façon à :
— avoir des cibles complémentaires comme par exemple supprimer les fichiers intermédiaires
— utiliser des variables
— donner des règles de construction des fichiers plus génériques
— pouvoir générer plusieurs exécutables distincts
— ...

4.1 Cibles complémentaires

En général, un fichier `makefile.txt` contient aussi des cibles complémentaires telles que :
— all : qui regroupe toutes les dépendances des exécutables à produire.
— clean : qui supprime tous fichiers intermédiaires générés
— mrproper : qui supprime tout ce qui peut être régénéré et permet une reconstruction complète du projet

Ajouter les deux cibles `clean` et `mrproper` à votre `makefile.txt`

4.2 Variables

L'utilisation de variables rend les évolutions bien plus simples et plus rapides. En effet plus besoin de changer l'ensemble des règles si le compilateur change, seule la variable correspondante est à modifier.

Une variable se déclare sous la forme NOM=VALEUR et s'utilise via $(NOM).
Les plus couramment définies sont :
— CC désigne le compilateur utilisée.
— CFLAGS regroupe les options de compilation
— LDFLAGS regroupe les options de l'édition de liens.
— EXEC donne le nom des exécutables à générer.

Modifier votre `makefile.txt` afin d'utiliser au mieux cette possibilité

Il existe aussi des variables pré-définies. Par exemple :

$@	Nom de la cible
$<	Nom de la première dépendance
$^	Liste des dépendances
$?	Liste des dépendances plus récentes que la cible
$*	Nom du fichier (sans son suffixe)

4.1 - Que fait cette cible ?

```
Func.o: Fonc.c
    $(CC) -o $@ -c $< $(CFLAGS)
```

4.2 - Modifier la cible `calculatrice` de la même manière

Il est possible de construire des règles génériques telle que :

```
%.o: %.c
    $(CC) -o $@ -c $< $(CFLAGS)
```

qui décrit comme construire un .o à partir d'un .c.

4.3 - Modifier votre fichier `makefile.txt` afin d'utiliser au mieux ces nouvelles possibilités

Enfin, il est possible de parcourir une liste quand celle-ci est donnée comme cible :

```
EXEC=p1 p2 p3
$(EXEC): %: %.c $(LDEP)
	$(CC) -o -$@ -$^
```

5 Serveur de calcul

Voila voila, vous avez maintenant tout savoir nécessaire pour répondre aux exigences du client et produire les versions finales V1a, V1b, V2a et V2b. Pensez à écrire le fichier `serveurCalculs.txt` permettant de construire les exécutables nécessaires à la version choisie par l'utilisateur (par exemple `make -f serveurCalculs.txt V1b`)