Projet Système

2009-2010

SimpleMake

Introduction

« make » est un logiciel traditionnel d'UNIX facilitant la compilation et l'édition de liens. Ce logiciel utilise un fichier de configuration que l'utilisateur crée, nommé Makefile (ou makefile), répondant à une syntaxe propre, déterminant les compilateurs à utiliser, les fichiers à compiler, les binaires à créer, etc...

L'objectif du projet était de créer un logiciel semblable à make – smake - , répondant au cahier des charges suivant :

« L'objectif de ce projet est d'écrire une version simplifiée de make(1) gràce à stat(2v), fork(2) et execvp(3). Vous devez fournir un Makefile dont la cible par défaut compilera votre projet sous le nom smake. Une partie de la note sera attribuée de manière semi-automatique par une batterie de tests : nous utiliserons votre programme pour compiler différents projets de test. Votre programme doit renvoyer un code d'erreur de 0 en cas de succès et seulement dans ce cas.

smake offre cependant moins de fonctionnalités que son grand frère : il ne permet que de compiler un seul programme C et suppose que vous utilisiez le compilateur gcc. La syntaxe de smake est la suivante : « smake [fichier-description] ». Si le nom du fichier de description est omis, on utilisera «smakefile». Si le fichier spécifié n'existe pas, smake termine avec un message d'erreur. »

C'est pour répondre à ce cahier des charges que nous avons effectué notre logiciel. Le sujet permettant un large choix de nommage, nous avons décidé d'appeler notre projet smake. (le sujet proposait également SimpleMake)

Difficultés

Au cours du développement de notre logiciel, nous avons été confronté à certaines difficultés, qu'elles soient dûes au sujet en lui-même, ou au principe de développement.

1. **Normalisation**

   Nous avons prêté une attention particulière au nommage de nos variables et de nos fonctions. Celles-ci correspondent aux critères habituels de codage.

   Parallèlement à cela, nous avons fait en sorte qu'il n'y ait pas de fuites mémoires. (Cf. valgrind) Chaque malloc entraine un free.

2. **Code d'erreurs**

   Dans un souci de débugging dans les cas où le projet à compiler n'entraine pas une sortie correcte (code 0), nous avons ajouté un ensemble de code d'erreurs :

0	sucessful
101	smakefile not found
202	more than one executable in the smakefile
303	no executable in the smakefile.
404	smakefile directory not found.
505	no source files in the smakefile.
606	header file missing.
707	source file missing.
808	exec failed.
909	exec failed.

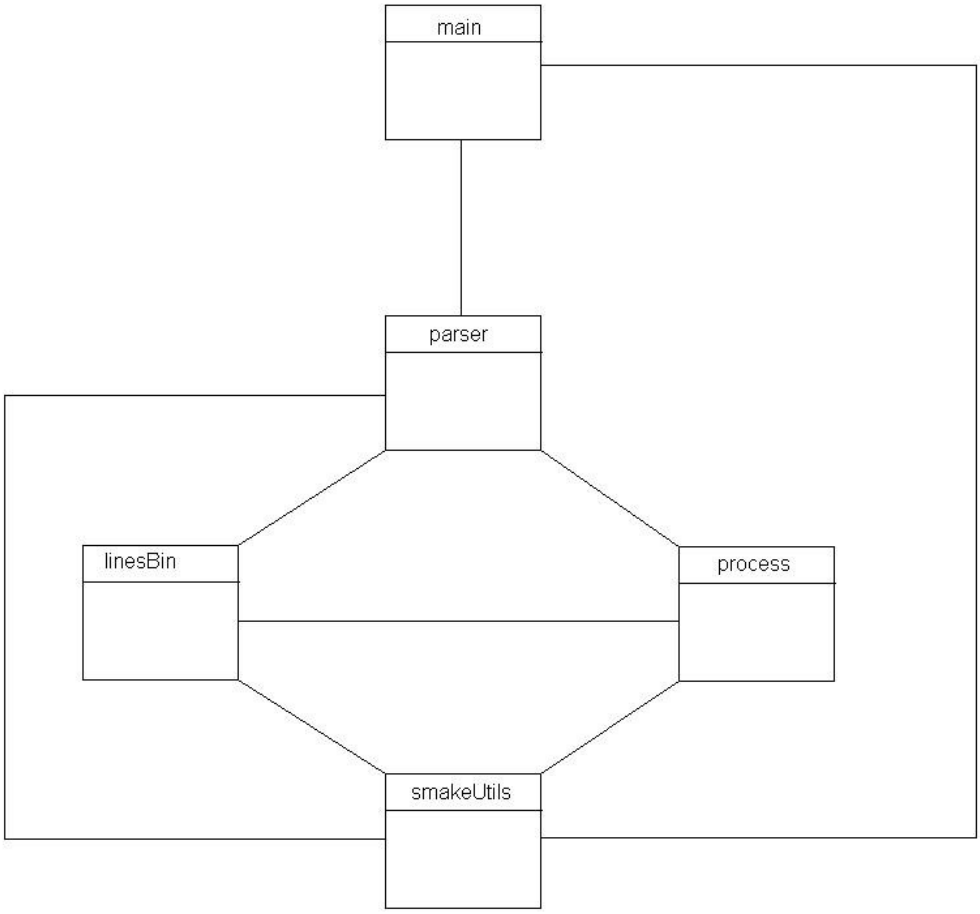
3. **Erreurs évitées**

   Il est également possible que le binaire « smake » ne se situe pas dans le même répertoire que le projet à compiler. Dans le cas où l'on ne tient pas compte de cela, il peut arriver que le programme renvoie une erreur, car il ne parviendra pas à trouver les fichiers *.c à compiler.

   Dans notre programme, nous utilisons la fonction « *chdir* ». Cela permet de régler des soucis qui peuvent survenir.

Architecture

Pour notre projet, nous avons choisi cette architecture logicielle :



main : Regarde les arguments et lance le programme.

Parser : Vérifie l'existence du smakefile, et sert à parcourir le makefile pour identifier les éléments.

LinesBin : définit la structure linesBin contenant les différentes données du smakefile, et gère la création, l'allocation, ainsi que la destruction de cette structure.

process : partie vérifiant les dates des fichiers .h et .o ; et lance la compilation si nécessaire.

smakeUtils : Utilitaires divers (printer).

Error.h : définition des valeurs constantes. (non-inclus dans le diagramme)

Temps passé

Pour ce projet, nous avons réparti également le temps passé.

(temps en heures)	Conception	Codage	Tests	Rapport
Matthieu	3	7	3	1
Christophe	3	6	3	2

Au final, notre projet nous a pris environ 14heures chacun.

Nous avons travaillé autour d'un repository svn (assembla), ce qui nous a permis de travailler facilement, avec toujours les versions mises à jour. De plus, cela permet de voir la durée du projet global. Le temps passé entre notre premier et notre dernier commit est de 54 jours.

Conclusion

A l'aide de ce projet, nous avons mieux compris le processus de compilation « make » très populaire sur le système UNIX. Nous pensons plus souvent utiliser cet outil lors de nos futurs projets.

Nous avons choisi de ne pas implémenter les fonctionnalités optionnelles, bien que nous ayons eu la réflexion suffisante pour savoir comment les implémenter : généraliser un peu notre structure de notre programme aurait permis l'ajout de la fonctionnalité du choix du compilateur (gcc /javac).

D'autre part, cela nous a permis de voir un peu plus en profondeur l'utilisation des makefile, que nous avions seulement survolé durant notre enseignement du langage C. Nous avons même utilisé notre programme pour tester la compilation de notre programme, ce qui a fonctionné.