[LC031]

Obtention d'un programme exécutable sous Unix

Rappel de généralités sur l'obtention d'un programme exécutable

Les trois sortes de langages de programmation

Il existe trois sortes de langages pour écrire un programme sur un ordinateur donné :

- le langage machine (binaire) : seul effectivement exécutable par le processeur (CPU), il dépend de celui-ci.
- le langage d'assemblage (mnémonique) : manière d'écrire le langage machine afin qu'il soit utilisable par l'homme ; il dépend évidemment aussi du processeur.
- les langages évolués (tels que Pascal, C, Fortran, Ada,...) : langages proche de l'homme, permettant d'écrire des programmes portables d'une machine à une autre.

Un programme exécutable est en fait un programme écrit en langage machine, mais il est en général obtenu comme résultat de la traduction d'un programme source écrit soit dans un (ou plusieurs) langage(s) de haut niveau, soit en langage d'assemblage, soit dans un mélange des deux. Cette traduction comprend plusieurs étapes :

La compilation

Réalisée par le compilateur, cette étape consiste à traduire un programme source (écrit en langage de haut niveau) en un programme objet (écrit en langage machine). Ce programme objet n'est cependant pas encore exécutable, car il lui manque un certain nombre d'informations, en particulier des adresses (le compilateur ne s'occupe pas de savoir à quelle(s) adresse(s) sera placée le programme dans la mémoire au moment de l'exécution).

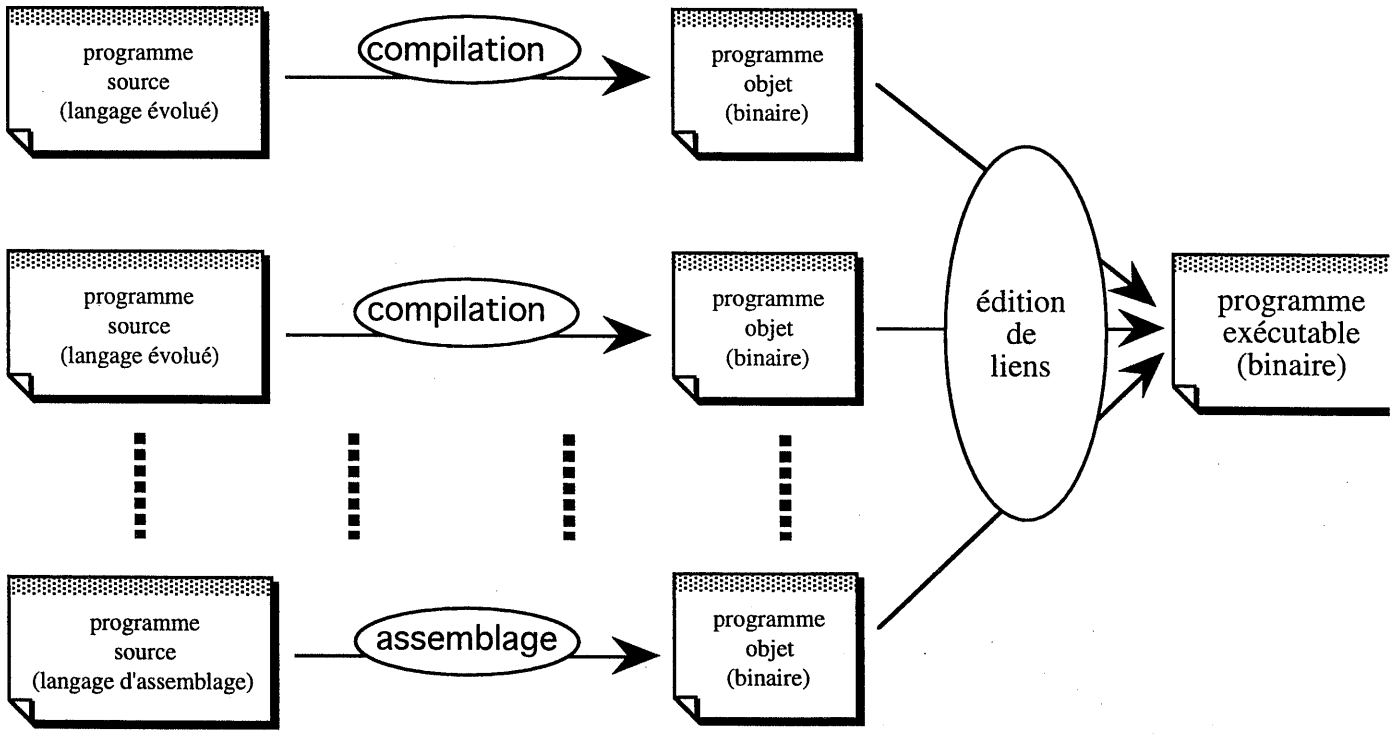
En outre, il est possible de fractionner un programme en plusieurs morceaux (ou modules) qui peuvent être éventuellement écrits dans des langages évolués différents, et que l'on pourra compiler séparément.

L'assemblage

Réalisée par l'assembleur, cette étape consiste à obtenir à partir d'un programme en langage d'assemblage, un programme en langage machine. Lorsqu'un programme est écrit en plusieurs modules, on peut en écrire certains en langage d'assemblage. Ce sont en général ceux que l'on veut optimiser pour qu'ils s'exécutent le plus rapidement possible.

L'édition de liens :

Réalisée par l'éditeurs de liens, cette étape consiste à obtenir (enfin) un programme exécutable à partir d'un ou plusieurs modules objets (ou bibliothèques de modules objets) compilés et assemblés séparément.



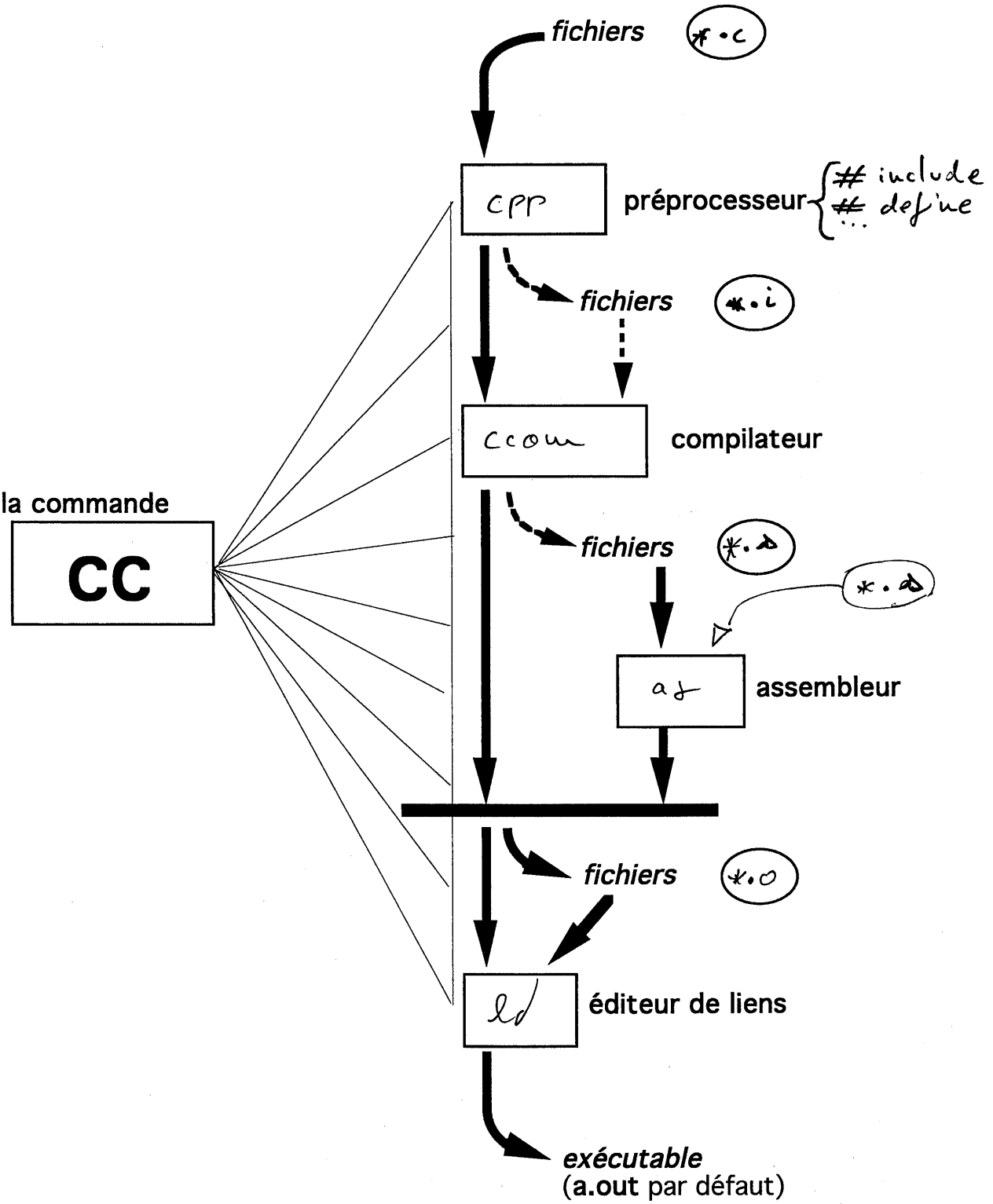
Qu'en est il sous Unix ?

Outils de compilations

On peut se contenter d'un seul utilitaire (dit compilateur) suivant le langage évolué que l'on utilise :

- **cc** pour le langage C
- **pc** pour le langage pascal
- **f77** pour le fortran
- etc...

En fait chacun de ces programmes, outre la compilation proprement dite, s'occupe de lancer d'autres outils pour le préprocessing (traduction des macros, inclusions des fichiers, etc.), l'assemblage éventuel et l'édition de liens.



Le schéma est analogue pour **pc** et **f77**. Par ailleurs il est bon de savoir qu'il existe d'autres compilateurs C que celui fourni de base par Unix, tels que par exemple le compilateur **gcc** du GNU (gratuit).

[LC021 p3]/[UX050 p4]

Syntaxe de la commande cc

pour plus de détails
[JMR] pages 256 à 259
man cc

cc [*-options*] *fich1 fich2 ...* [**-o** *fichexec*] [**-l***library*]

fich1, *fich2*,... sont les noms des fichiers sources à prendre en compte. Les terminaisons de ces noms indiquent les types de ces fichiers :

- **.c** fichiers sources écrit en C.
- **.i** fichiers sources écrit en C. Ces fichiers sont supposés être le résultat d'un 'préprocessing' précédent (voir option **-P**). Le préprocesseur **cpp** ne leur sera donc pas appliqué.
- **.s** fichiers sources écrits en langage d'assemblage.
- **.o** fichiers objets relogeables destinés à être liés par le '**linker**' (éditeur de liens). Ces fichiers sont normalement le résultat d'une compilation précédente pour laquelle la phase d'édition de liens a été supprimée (cf option **-c** ci-dessous).

quelques options :

- **-Aa** suivre la norme ANSI (en particulier style moderne pour les fonctions).
- **-Ae** idem en norme ANSI étendue (en particulier accepte commentaires //)
- **-g** génère des informations pour les débogueurs.
- **-O +O** **+O1 +O2 +O3** diverses options d'optimisations. (Elles sont incompatibles avec l'option -g).
- **-o** *fichexec* permet de préciser le nom du fichier exécutable (par défaut : **a.out**).
- **-l***library* indique une (ou plusieurs) bibliothèque(s) de fonctions à prendre en compte en plus de la bibliothèque standard. Cette option doit être en fin de ligne de commande.[i]

  exemple : **-lm** → utiliser bibliothèque de fonctions math. (fichier libm.a ou libm.so ou libm.sl)
- **-L***répert* indique un (ou plusieurs) répertoire(s) où **l'éditeur de liens** doit aller chercher le(s) bibliothèque(s) de fonctions.[i]
- **-I***répert* indique un (ou plusieurs) répertoire(s) où le préprocesseur doit aller chercher le(s) fichier(s) à inclure par **#include<...>**[ii]
- **-D***identif*
  -D*identif=valeur* simule un **#define.** (option transmise au préprocesseur)
- **-c** supprime la phase **'édition de liens'** → fichier **.o** pour chaque fichier source (**.c** **.i** ou **.s**).
- **-P** arrête la compilation juste après la phase de 'preprocessing' effectuée par le préprocesseur **cpp**; chaque fichier **.c** donne alors un fichier **.i** correspondant.
- **-v** mode **verbeux**. Donne une description détaillée de la compilation.

[i] Cette option est en fait transmise à l'éditeur de liens.
[ii] Cette option est en fait transmise au préprocesseur.

Exemples :

- pour compiler un programme écrit en un seul fichier source prog.c écrit en suivant la norme ANSI et l'on souhaite obtenir un exécutable appelé prog :

```
cc -Aa prog.c -o prog
```

- pour compiler un programme écrit en trois modules C (appelés prog.c, prog1.c, prog2.c) suivant la norme ANSI étendue, plus un module assembleur proga.s, en utilisant la bibliothèque des fonctions mathématiques et en souhaitant obtenir un exécutable comm:

```
cc -Ae prog.c prog1.c prog2.c proga.s -o comm -lm
```

ou bien

```
cc -Ae prog*.c proga.s -o comm -lm
```

- si en outre, l'un des fichiers sources prog1.c (par exemple) avait été compilé précédemment, et que le fichier assembleur avait été assemblé, il suffit de faire :

```
cc -Ae prog.c prog1.o prog2.c proga.o -o comm -lm
```

Autres outils

L'utilitaire **lint** permet de faire une analyse syntaxique. Il ne génère aucun code, mais vérifie la syntaxe de vos fichiers sources et signale quelques erreurs du type avertissements (warnings) qui ne sont pas détectés par **cc**. La syntaxe est en gros celle d'un appel à **cc**. Si on utilise **gcc** celui-ci indique directement les avertissements.

Le débogage se fait avec **xdb** ou **gdb** suivant le compilateur qu'on a utilisé.

On dispose aussi de commandes de traçage qui insèrent des lignes supplémentaires dans les fichiers sources, permettant ainsi d'obtenir à l'exécution des informations sur les appels de fonctions (**ctrace**), sur les appels systèmes et les réceptions de signaux (**truss**) ou d'obtenir un graphe des appels de fonctions (**cflow**).

La commande **make** permet de mettre à jour un programme, surtout lorsqu'il est écrit en plusieurs module. C'est-à-dire de ne recompiler que ce qui a été modifié. Elle se sert pour cela d'un fichier appelé **makefile** qui doit être dans le répertoire de travail. Ce répertoire ne devrait contenir que les fichiers du programme sur lequel on travaille (sources, objets, exécutables et *makefile*). Vous pouvez recopier le fichier *Makefile* qui se trouve dans /users/tpinfo/blanc et le modifier afin que **make** compile vos programmes (juste le nom du programme à modifier à chaque fois). En voici des extraits :

```
# Makefile simple pour compilation de programmes C - JPBlanc - IUT Informatique de Clermont I -
# (ce makefile est utilisable même si vous ne connaissez pas la commande make)
...
# Nom du programme exécutable après compilation (remplacez
# "monprog" dans la ligne suivante par le nom de votre programme)
PROG=monprog
...
# Option(s) pour norme ANSI (décommentez uniquement la ligne qui convient).
#CFLAGS=                # pas de norme ANSI
#CFLAGS= -Aa -g         # norme ANSI
CFLAGS= -Ae -g          # norme ANSI étendue (avec les //)
...
# Quelle(s) bibliothèque(s) faut il utiliser. Par exemple '-lm' signifie
# bibliotheque  'libm.sl' c'est-à-dire biblio des fonctions mathématiques
LIBS= -lm

# Objets : Si vous avez plusieurs fichiers .c rajoutez ici les .o correspondants
#              à la suite de $(PROG).o
OBJS=$(PROG).o
#------------------------------------------------------------------------------ règles
$(PROG): $(OBJS)              #règle pour édition de lien de votre programme
  cc  $(OBJS) -o $(PROG) $(LIBS)

.c.o:                         #règle d'inférence pour compilation
  cc $(CFLAGS)  $(DFLAGS) -c $<

.c:                           #règle d'inférence pour compilation avec édition de liens
  cc $(CFLAGS)  $(DFLAGS)  $< -o $@  $(LIBS)

rebuild: clean $(PROG)        #règle pour recompilation complète de votre programme

clean:                        #règle pour nettoyage (suppression des fichiers objets).
  rm $(OBJS)
```