CHAPITRE II : CHEMINEMENT D'UN PROGRAMME DANS UN SYSTEME D'EXPLOITATION

2.1 CHAINE DE PREPARATION D'UN PROGRAMME :

Le développement d'un programme, depuis l'analyse du problème jusqu'à sa mise au point, nécessite de nombreux outils logiciels qui constituent un environnement de programmation. Pour fonctionner, ces outils utilisent les services du système d'exploitation. La figure suivante illustre un environnement de programmation contenant un nombre minimum d'outils classiques.

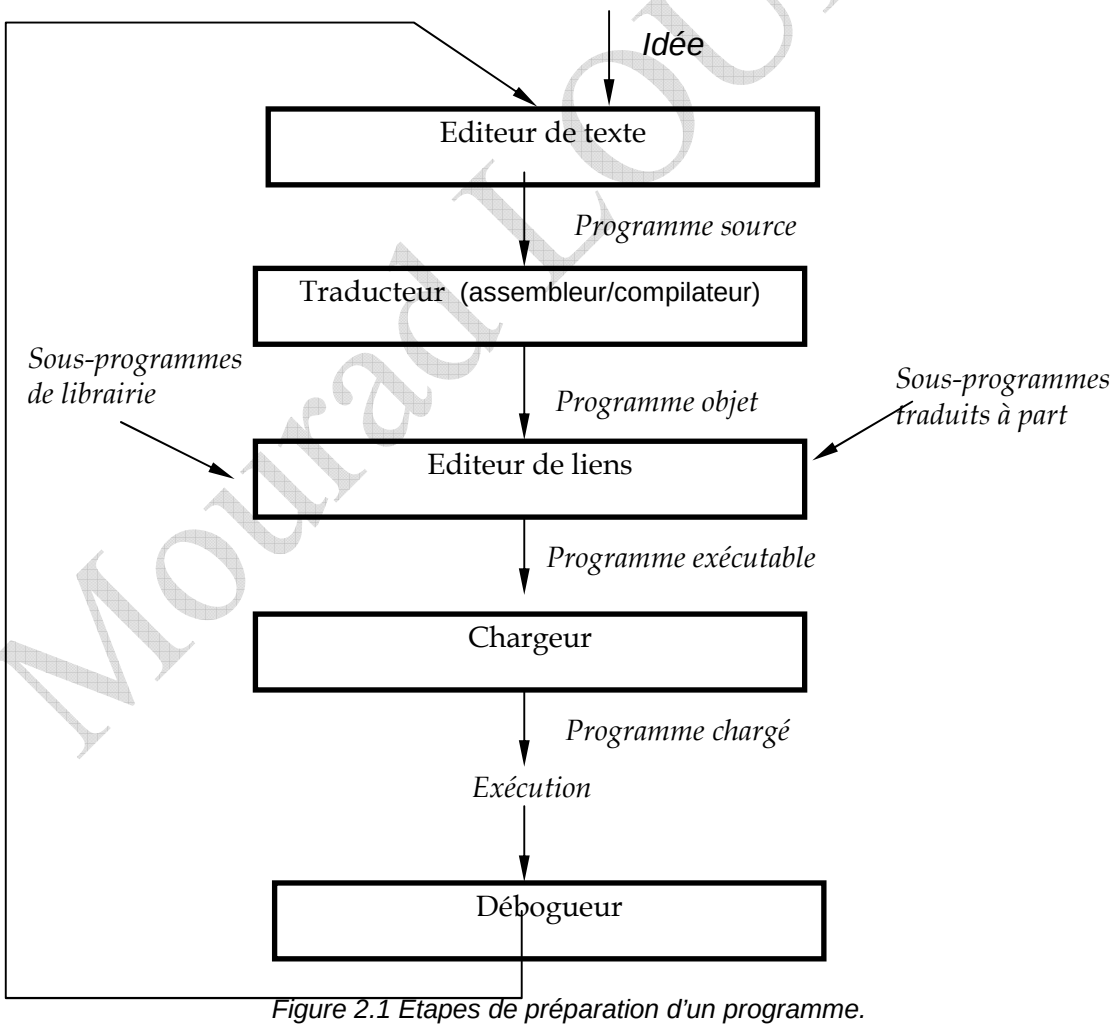

Figure 2.1 Etapes de préparation d'un programme.

L'éditeur de textes : Un éditeur de texte (text editor) est un logiciel interactif qui permet de saisir du texte à partir du clavier et le stocker dans un fichier.

Le Traducteur : Le traducteur est un logiciel qui, traduit un programme source écrit en langage de haut niveau en un programme objet.

L'éditeur de textes : Un éditeur de liens (linker) est un logiciel qui permet de combiner plusieurs programmes objet en un seul programme exécutable.

Le Chargeur : Le programme exécutable, obtenu après l'édition de liens, doit être chargé en mémoire centrale pour être exécuté. Le chargeur (loader) s'occupe de cette tâche.

Le débogueur : Le débogueur (debugger) est un logiciel qui facilite la mise au point de programmes (détection et correction des erreurs, ou bugs). Il permet d'examiner le contenu de la mémoire ainsi que le contenu des différents registres. Ainsi, on peut suivre l'exécution pas à pas, c'est à dire instruction après instruction.

2.2 LA TRADUCTION D'UN PROGRAMME :

Pour l'ordinateur, un programme écrit dans un langage qui n'est pas le sien est en fait une simple suite de caractères plus ou moins longue. Il doit être traduit par un traducteur . Le traducteur peut se présenter sous l'une des deux formes suivantes:

- L'***interpréteur*** parcourt en permanence la suite de caractères, analyse les actions définies par le programme, exprimées dans le langage évolué, et exécute immédiatement la séquence d'action de la machine qui produit les mêmes effets.
- Le ***compilateur*** parcourt la suite de caractères, analyse les actions du programme et la traduit en une suite d'instructions dans le langage de la machine.

Quels que soient le langage et la méthode de traduction utilisés, le programme est toujours une suite de caractères. C'est ce que nous appellerons un *programme source*. Le traducteur doit analyser cette suite de caractères pour retrouver les constituants essentiels, vérifier la conformité avec la définition du langage, et en déduire la sémantique. Un interpréteur doit ensuite exécuter les actions correspondant à cette sémantique, alors qu'un assembleur ou un compilateur doit traduire cette sémantique dans un autre langage. On appelle *programme objet*, le résultat de cette traduction.

2.2.1 L'analyse lexicale

L'analyse lexicale a pour but de segmenter le programme source et en tirer tous les items lexicaux (constantes, variables, mots clés, …).

Exemple : soit le programme suivant .

```
begin x := 23; end
```

Cette suite de caractères doit être découpée de la façon suivante:

Begin	x	:=	23	;	**end**

2.2.2 L'analyse syntaxique

Le résultat obtenu après analyse lexicale est une suite de symboles. Mais, toute suite de symboles ne constitue pas un programme. Il faut donc vérifier la concordance de la suite avec la structure du langage, sans se préoccuper, pour le moment, de la sémantique du programme. C'est ce que l'on appelle l'*analyse syntaxique*.

La plupart des langages de programmation modernes définissent une syntaxe du langage par des *règles de production*, qui décrivent la façon dont on peut construire une suite de symboles pour qu'elle constitue un programme. Le programmeur construit alors son programme en appliquant ces règles de façon répétitive. L'analyse syntaxique consiste à retrouver, à partir du programme source, ou en fait à partir de la suite de symboles, quelles sont les règles que le programmeur a appliquées.

2.2.3 L'analyse sémantique

Après avoir contrôlé la correction du programme, et en avoir obtenu la structure syntaxique, la traduction continue par la phase d'*analyse sémantique*, qui a pour but de trouver le sens des différentes actions voulues par le programmeur. Trois problèmes doivent alors être résolus:

- quels sont les objets manipulés par le programme,
- quelles sont les propriétés de ces objets,
- quelles sont les actions du programme sur ces objets.

2.2.4 La génération et l'optimisation de code

La phase de génération de code est la phase de traduction proprement dite. Elle n'est entreprise que si aucune erreur n'a été trouvée. Elle consiste à construire un programme équivalent à la sémantique obtenue dans la phase précédente, dans un nouveau langage, et donc en particulier dans le langage de la machine lui-même.

2.3 L'EDITION DE LIENS :

Les programmes relativement complexes ne peuvent être écrits entièrement dans un seul fichier, il est nécessaire de pouvoir les découper en morceaux, chacun d'eux pouvant alors être modifié, compilé et testé séparément des autres. Les morceaux doivent ensuite pouvoir être rassemblés pour former le programme complet. Nous avons appelé *module source* un tel morceau de programme pouvant être traité de façon indépendante par un traducteur, et *module objet* le résultat de cette traduction. Nous nous intéressons maintenant à la reconstruction du programme complet à partir des modules objets. Nous avons appelé *édition de liens* cette opération. Elle est réalisée par un programme spécifique, appelé *éditeur de liens* (en Anglais *link editor*).

2.3.1 La notion de module translatable

Les traducteurs séparent souvent l'espace mémoire du module en plusieurs *sections* suivant la nature de leur contenu, comme par exemple:

- code des instructions,
- données constantes,
- données variables.

Ces différentes sections, gérées séparément par le traducteur peuvent être mises n'importe où en mémoire. Les informations de placement produites par le traducteur précisent alors pour chaque emplacement qui contient une adresse, la section à laquelle cette adresse se réfère. On dit alors que le module objet est *translatable*. L'opération de translation consiste à ajouter à chacun des emplacements qui contient une adresse, la valeur de l'adresse effective d'implantation finale de la section.

2.3.2 La notion de lien

La construction d'un programme à partir d'un ensemble de modules n'est pas simplement la juxtaposition en mémoire de l'ensemble de ces modules. Il est nécessaire d'assurer une certaine forme de coopération entre les modules, et donc qu'ils puissent communiquer. Cette communication peut se faire par *variable globale*, c'est-à-dire qu'une variable appartenant à un module est accessible depuis un ou plusieurs autres modules. Elle peut se faire par *appel de procédure*, c'est-à-dire qu'une procédure ou une fonction appartenant à un module est accessible depuis un ou plusieurs autres modules. On appelle *lien* l'objet relais, qui permet à un module d'accéder à un objet appartenant à un autre module. L'établissement de ce lien passe par un intermédiaire, le *nom* du lien, permettant de le désigner dans les deux modules. Ce nom est en général une chaîne de caractères. Nous appellerons *lien à satisfaire* la partie du lien située chez l'accédant, et *lien utilisable* la partie située chez l'accédé.

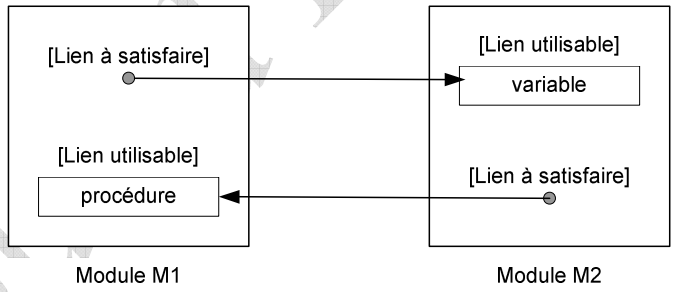

Fig. 2.6 La notion de lien.

A titre d'exemple, considérons les deux programmes en langage C suivants : prog1.c et prog2.c.

Prog1.c

```
extern int v1 ;
extern int proc1(int x) ;
float v2 ;

main()
{
      int a, x ;
      x = v1 + proc1(a) ;
}
```

Prog2.c

```
int v1 ;
extern float v2 ;
int proc1(int x) ;

main()
{
      float b ;
      b = v2 *2;
}

int proc1(int x)
{
      int y ;
      return (x * y /2) ;
}
```

On remarque dans cet exemple que pour le programme *Prog1.c* les liens à satisfaire sont la variable *v1* et la procédure *proc1*, alors que le lien utilisable est la variable *v2*. par contre pour *Prog2.c*, le lien à satisfaire est la variable *v2* alors que les liens utilisables sont la variable *v1* et la procédure *proc1*.

Module	**Liens à satisfaire**	**Liens utilisables**
Prog1.c	Variable v1 Procédure Proc1	Variable v2
Prog2.c	Variable v2	Variable v1 Procédure proc1