Les systèmes d'exploitation

Wadoud Bousdira[1]
wadoud.bousdira@univ-orleans.fr

[1]LIFO, University of Orléans
Orléans, France

Orléans, 2023

Cheminement d'un programme dans un système

Exécuter un programme nécessite de nombreux outils utilisant les services du SE

- ces outils facilitent le développement de projets,
- et forment la chaîne de production de programmes

Les plus répandus :

- le compilateur / traducteur,
- l'éditeur de liens,
- le chargeur
- l'éditeur de textes
- le débogueur.

Programme source : vu comme une suite de caractères.

Cheminement d'un programme dans un système

langage de haut niveau

allocation_memoire.c (/tmp) - gedit

Fichier Édition Affichage Rechercher Outils Documents Aide

Nouveau Ouvrir Enregistrer Imprimer... Annuler Rétablir Couper Copier Coller Rechercher Rempla

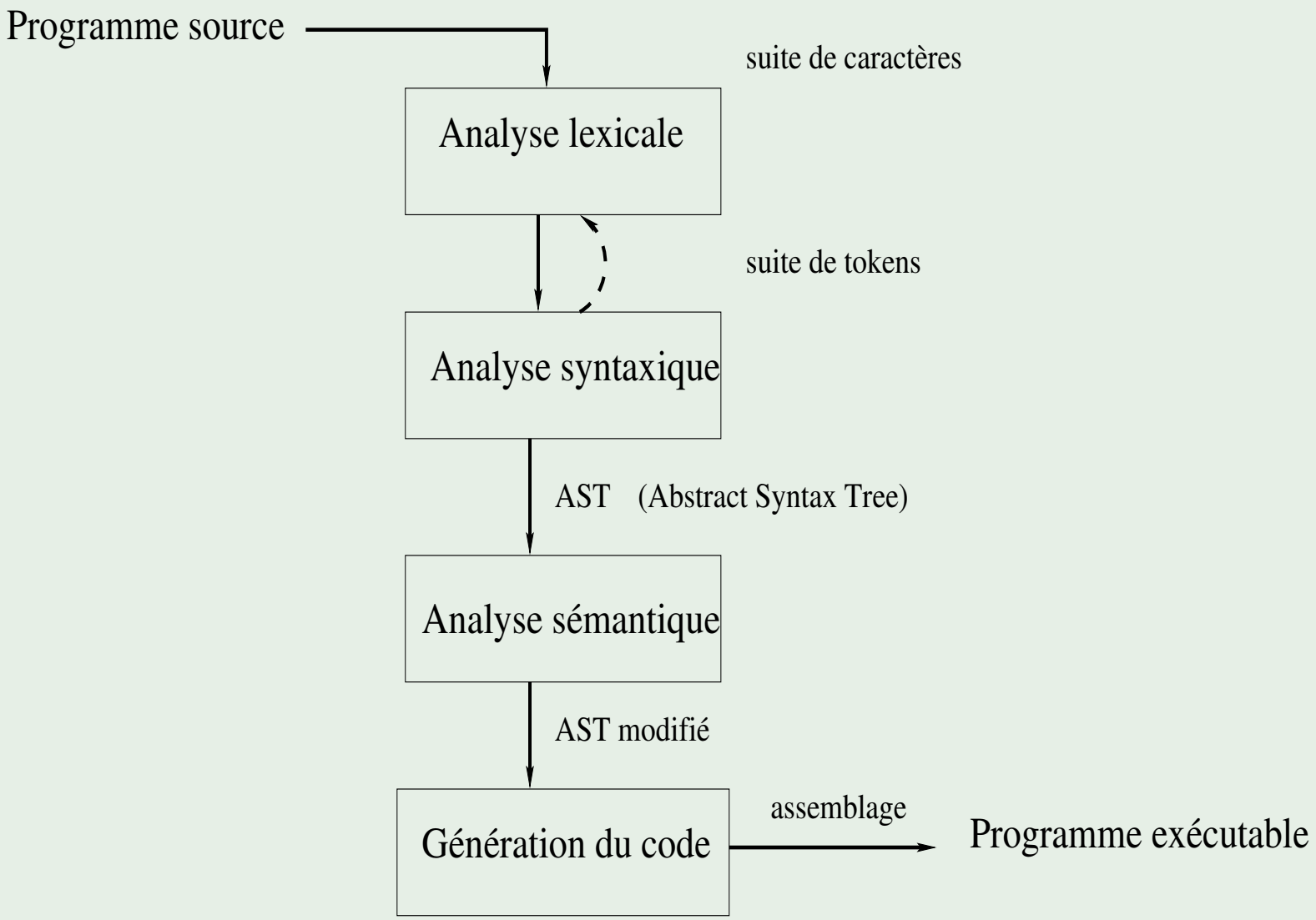
allocation_memoire.c

```c
/* La gestion de la mémoire n'est pas intégrée au langage
   mais assurée par des fonctions de la bibliothèque standard. */
#include <stdlib.h>

struct int_list {
    struct int_list *next; /* pointeur sur l'élément suivant */
    int value;             /* valeur de l'élément */
};

/*
 * Ajouter un élément à la suite d'un autre.
 * node : élément après lequel ajouter le nouveau
 * value : valeur de l'élément à ajouter
 * Retourne : adresse de l'élément ajouté, ou NULL en cas d'erreur.
 */
struct int_list *insert_next(struct int_list *node, int value) {
    /* Allocation de la mémoire pour un nouvel élément. */
    struct int_list *const new_next = malloc(sizeof(struct int_list));

    /* Si l'allocation a réussi, alors insèrer new_next entre node
       et node->next. */
    if (new_next) {
        new_next->next = node->next;
        node->next = new_next;
        new_next->value = value;
    }

    return new_next;
}
```

langage binaire

```
0000000 0000 0001 0001 1010 0010 0001 0004
0000010 0000 0016 0000 0028 0000 0010 0000
0000020 0000 0001 0004 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000
0000040 0004 8384 0084 c7c8 00c8 4748 0048
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028
0000060 00fc 1819 0019 9898 0098 d9d8 00d8
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c
0000080 8888 8888 8888 8888 288e be88 8888
0000090 3b83 5788 8888 8888 7667 778e 8828
00000a0 d61f 7abd 8818 8888 467c 585f 8814
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd
00000c0 8a18 880c e841 c988 b328 6871 688e
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888
0000100 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

Cheminement d'un programme dans un système



Cheminement d'un programme dans un système

- Langage machine : constitué de bits, impraticable.

⇒ remplacer les codes binaires des opérations de base par des codes mnémoniques (tokens)
Ex. ADD, DIV, SUB, MOV, JUMP, etc... plus faciles.

Assemblage

retrouver les codes binaires correspondant à ces noms abrégés.

- langage d'assemblage : difficile à mettre en œuvre, fortement dépendant de la machine.
- assembleur : programme qui traduit un programme écrit en langage d'assemblage, à base de codes mnémoniques d'opérations d'une machine dans le langage binaire de cette machine.

langage d'assemblage

```
; trouve n!
segment .text
      global _fact
_fact:
      enter  0,0

      mov    eax, [ebp+8]     ; eax = n
      cmp    eax, 1
      jbe    term_cond        ; si n <= 1, terminé
      dec    eax
      push   eax
      call   _fact            ; appel fact(n-1) récursivement
      pop    ecx              ; réponse dans eax
      mul    dword [ebp+8]    ; edx:eax = eax * [ebp+8]
      jmp    short end_fact
term_cond:
      mov    eax, 1
end_fact:
      leave
      ret
```

FIG. 4.15 – Fonction factorielle récursive

Cheminement d'un programme dans un système

Si langage évolué, le programme spécialisé chargé de la traduction peut se présenter comme

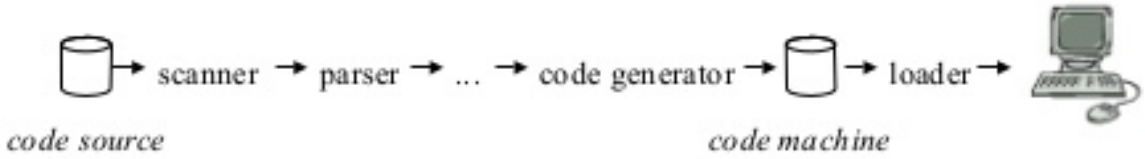
- un interpréteur : c'est un programme qui simule une machine qui comprend un langage évolué
  - parcourt en permanence la suite de caractères
  - analyse les actions définies par le programme, exprimées dans le langage évolué
  - exécute immédiatement la séquence d'actions de la machine qui produit les mêmes effets.
- un compilateur : c'est un programme qui traduit un programme écrit dans un langage évolué dans le langage binaire d'une machine, pour que celle-ci puisse l'exécuter
  - parcourt la suite de caractères
  - analyse les actions du programme
  - traduit le programme en une suite d'instructions dans le langage machine.

Schémas d'exécution d'un programme

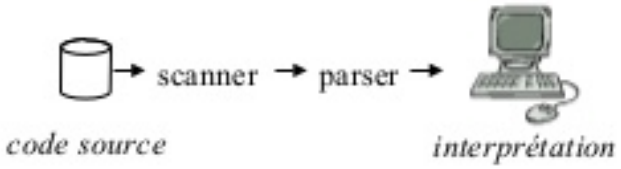
Schématiquement

Différence entre Compilateur et Interpréteur

Compilateur Traduit vers le code machine

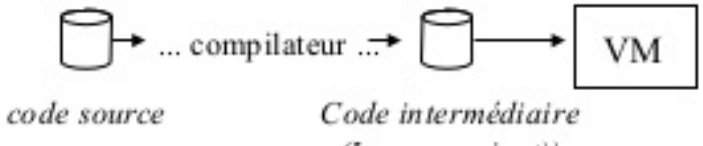


Interpréteur exécute le code source "directement"



- Les instructions d'une boucle sont scannées et analysées à chaque itération

Variante: interprétation du code intermédiaire



- *Le code source* est traduit dans le code d'une *machine virtuelle* (VM)
- VM interprète le code

Schémas d'exécution d'un programme

Exemple :

- compilation : C, C++, Fortran, Ada
- Interprétation : langage machine, shell Unix, PostScript
- l'un ou l'autre : Lisp, Caml
- la solution hybride : le langage JAVA.

Mécanisme d'exécution des programmes

Comparaison compilation-interprétation

	Compilation	Interprétation
Efficacité	- le code engendré s'exécute directement sur la machine physique - ce code peut être optimisé.	- l'interprétation directe est souvent longue - pas de gains sur les boucles
Mise au point	pas toujours facile de relier une erreur d'exécution au texte source.	- lien direct entre instruction et exécution - possibilités étendues d'observation et trace intégrées
Cycle de modification-réexecution	toute modification du texte impose de refaire le cycle complet (compilation, édition de liens, exécution)	cycle très court modification et réexecution

Schéma mixte d'exécution

Objectifs :

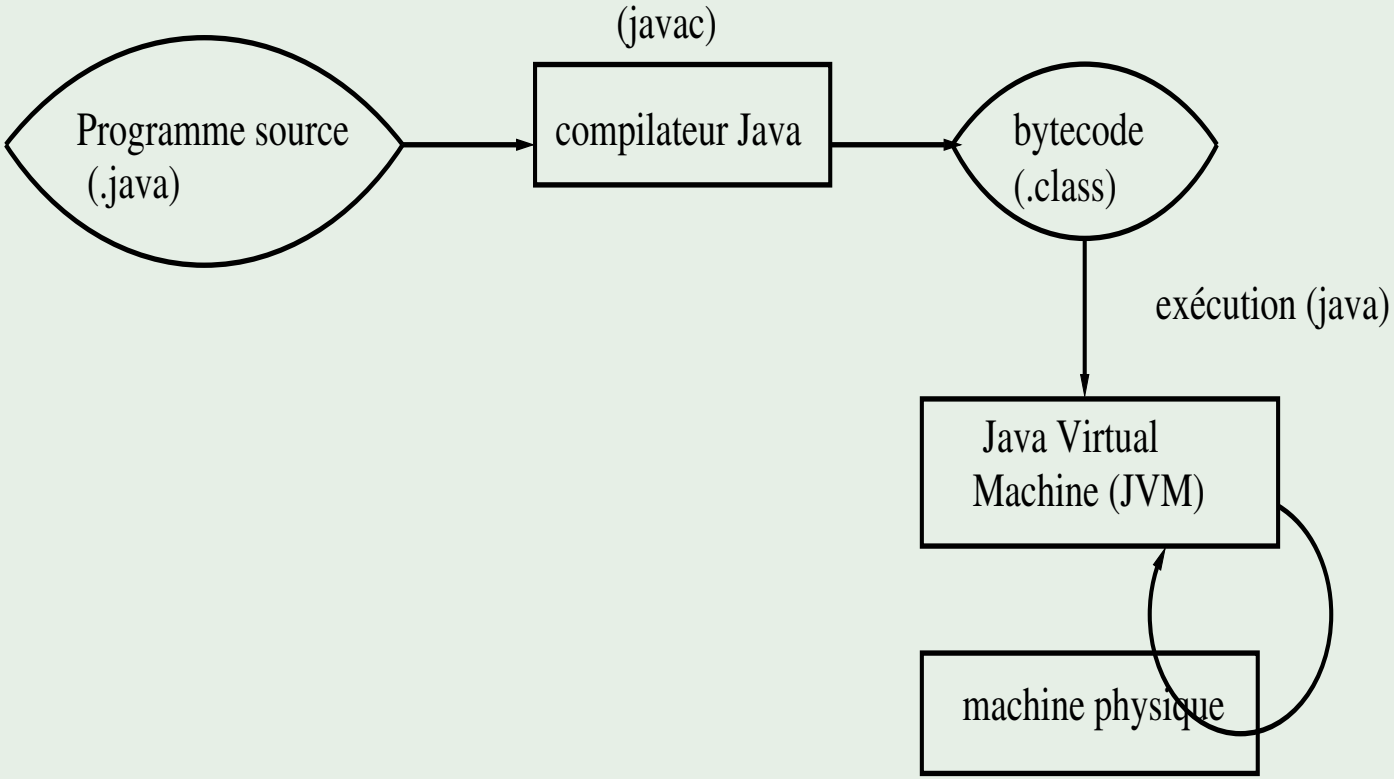
- tenter de combiner les avantages des schémas de compilation et d'interprétation,
- améliorer la portabilité entre machines via un langage intermédiaire standard.

Principes :

- définir un langage intermédiaire et une machine virtuelle capable d'interpréter ce langage,
- écrire un compilateur initial (source) vers le langage intermédiaire,
- écrire un interprète du langage intermédiaire (i.e. un simulateur de la machine virtuelle).

Schéma mixte d'exécution

Exemple : Java



- La phase de compilation est indépendante de la machine physique
- Portage sur une nouvelle machine = réécriture de la JVM sur cette machine.

Mécanisme d'exécution des programmes

Compilateur et interpréteur comportent trois phases d'analyse :

1. analyse lexicale : reconnaît dans une chaîne de caractères la suite de symboles du langage et en donne une codification interne.
   - Automates d'états finis.
2. analyse syntaxique : retrouve la structure syntaxique d'une suite de symboles et vérifie sa conformité / règles du langage.
   - Grammaires non contextuelles, LR(1), LALR(1).
3. analyse sémantique : trouve une signification aux actions définies par le programme.
   - Grammaires attribuées, arbre de syntaxe abstrait décoré.

Mécanisme d'exécution des programmes

- L'interpréteur exécute les actions correspondantes à cette sémantique.
- L'assembleur (ou compilateur) traduit cette sémantique dans un autre langage $\rightsquigarrow$ programme objet.

Découpage du programme en morceaux, traduits séparément, rassemblés par l'éditeur de liens.

Impossible avec un interpréteur.

Mécanisme d'exécution des programmes

Un programme est

- exécutable sur une machine constituée d'une mémoire adressable qui contient ses instructions et ses données,
- composé d'instructions que le processeur est capable d'interpréter. Instructions et données sont contenus dans des segments distincts.

Un segment

ensemble d'informations désignable et manipulable comme un tout, qui occupe en mémoire une suite d'emplacements contigus.

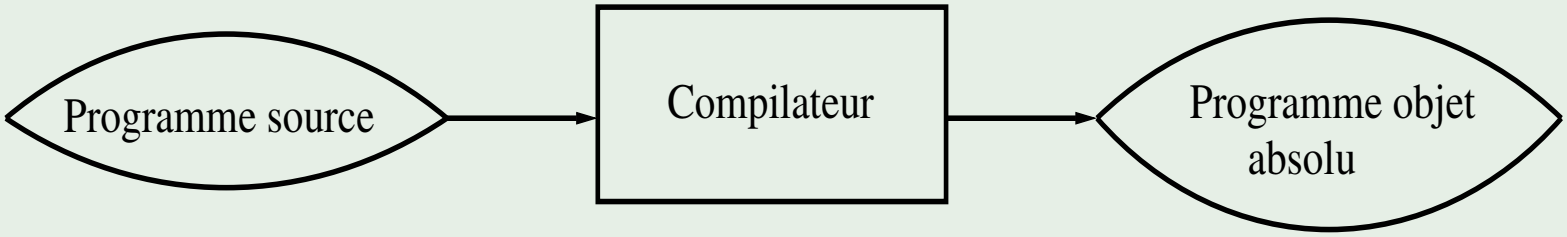
Mécanisme d'exécution des programmes

- L'état de la machine est défini en des points observables (évolution discrète) par
  - l'état du processeur
  - et l'état de la mémoire.
- L'état de la mémoire est défini par le contenu des segments qui y sont chargés.
- L'état du processeur est défini par le contenu d'un ensemble de registres programmables ou internes.

Schéma d'exécution d'un programme

Cycle de vie d'un programme compilé

- Compilation vers un programme objet absolu
  Les adresses sont fixes en mémoire.



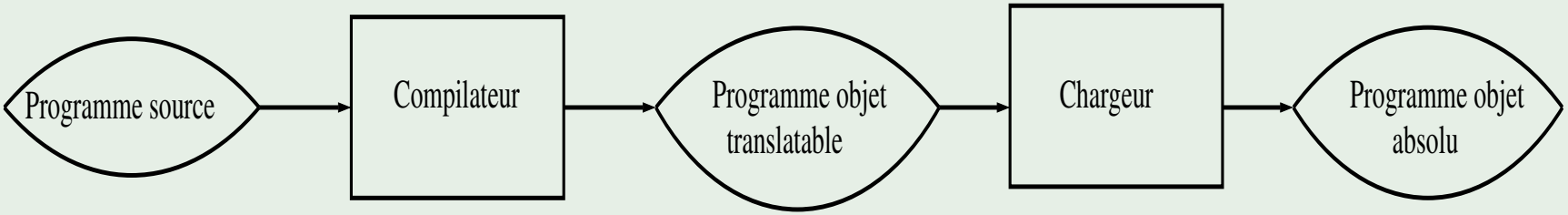
Contrainte : le programme ne peut pas être déplacé en mémoire (par ex. pour le combiner avec d'autres).
Chaque objet est désigné par l'adresse absolue de l'emplacement qui le contient. Un tel programme est

- immédiatement exécutable,
- toute modification est difficile, comporte un risque élevé d'erreur.

Schéma d'exécution d'un programme

Cycle de vie d'un programme compilé

- Compilation vers un programme objet translatable
  Les adresses sont définies à une translation près. Les identificateurs sont remplacés par des adresses relatives à l'origine du programme.



  - Les adresses sont relatives au début du programme ⇒ le programme n'est pas exécutable en l'état.
  - Nécessité de passer par un chargeur pour rendre le programme exécutable.

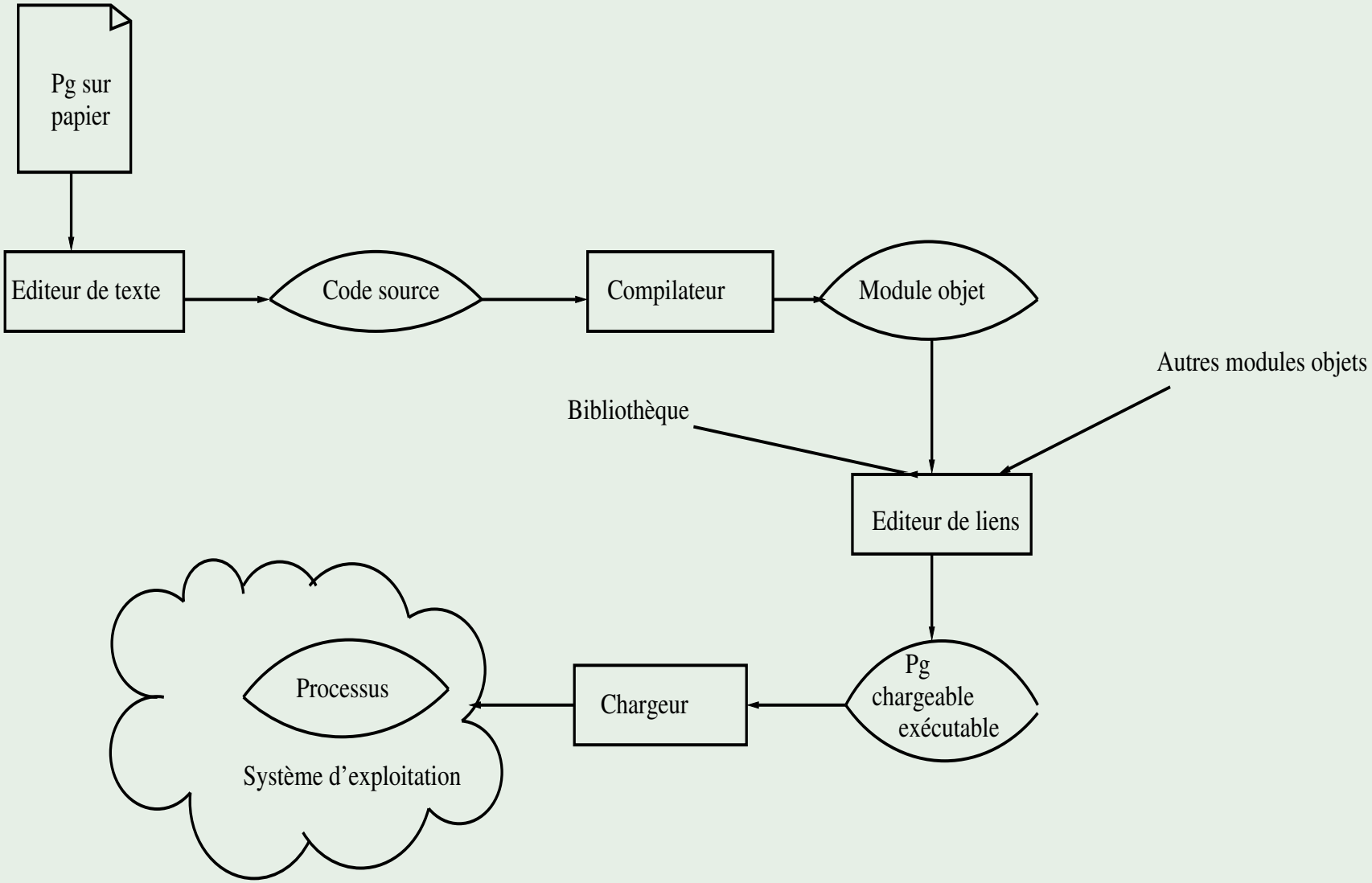
Schéma d'exécution d'un programme

Exemples

- `gcc -c prog.c`
  produit un pg objet translatable dans le fichier `prog.o`
- `gcc -o prog prog.c`
  produit un pg objet objet absolu dans le fichier `prog` (appelle le compilateur et le chargeur).

Schéma d'exécution d'un programme

Schéma général



Le chargement

Son rôle

mettre sous forme absolue un module objet translatable (ou relogeable).

- prend pour paramètre, l'adresse absolue A où est chargé le module
- à chaque adresse est associée une marque indiquant si elle est absolue ou translatable.
  (en pratique, les seuls objets en absolu désignent des constantes.)
  L'indicateur est placé dans la phase de traduction.
- procède par substitution : toute adresse relative a est remplacée par l'adresse $a + A$

L'édition de liens

Son rôle

rassembler les modules traduits séparément en un module unique : le module chargeable. Ce module est translatable ou absolu selon que l'éditeur de liens effectue aussi le chargement ou non

- les bibliothèques,
- les différents modules objets construits par compilation séparée.

Un programme ne s'exécute quasi jamais seul !

Module :

Unité logique du programme (procédure, programme principal,...)

L'éditeur de liens

Nécessite deux passes :

- Passe 1 : détermine l'adresse absolue de tous les objets externes.
  Construit une table globale des identificateurs externes *tg_ex*, qui contient pour chaque objet externe, le couple $(id, @abs(id))$
- Passe 2 : résout la référence à ces objets.
  effectue la translation des adresses et le chargement en mémoire, en utilisant la table *tg_ex*

L'éditeur de liens

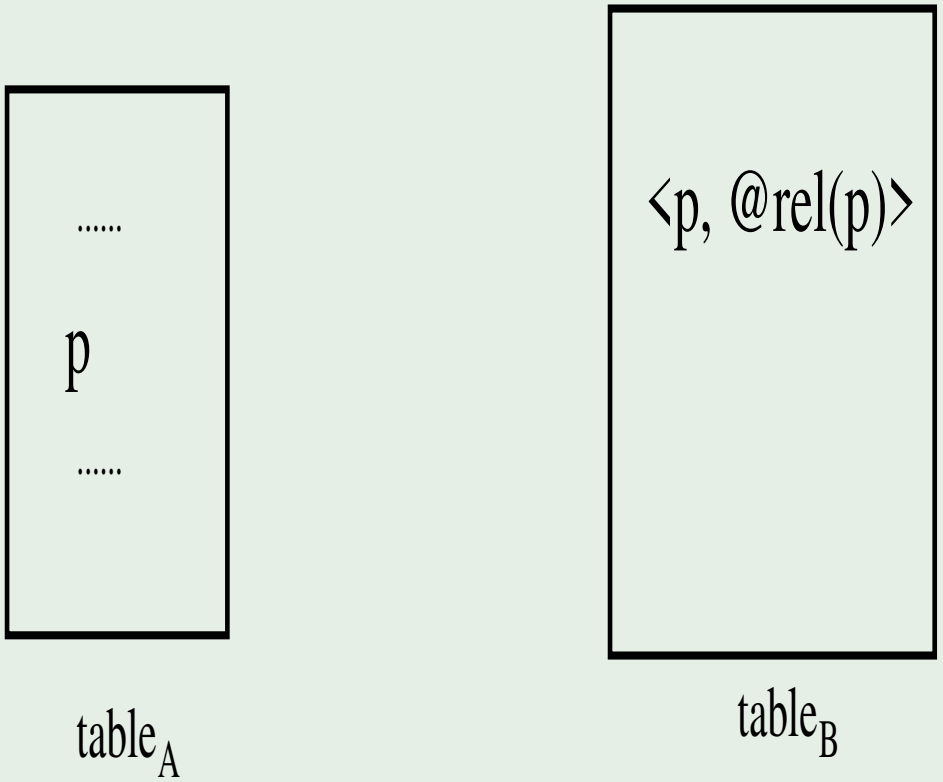
Pourquoi deux passes ?

- le module A utilise un identificateur p défini dans le module B,
- A et B sont compilés séparément.
- lors de la compilation de A, il est impossible d'associer une adresse à p tant qu'on ne connaît pas B.

L'éditeur de liens

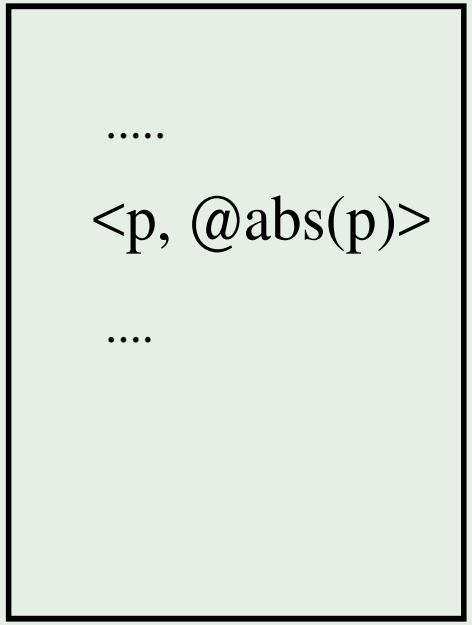
A fait référence à des objets définis dans B (ex. p)

- Avant la passe 1 :
  - La table table_A désigne les références externes de A (objets non définis dans A et utilisés par A),
  - la table table_B contient les définitions externes de B (définis dans B et utilisés à l'extérieur),



L'éditeur de liens

- Après la passe 1 : table globale *tg_ex* des définitions externes :



- Après la passe 2 :
  À toute instruction dans A utilisant p, l'éditeur de liens substituera l'adresse de p par $@abs(p)$

L'éditeur de liens

- À la fin de la passe 2, l'éditeur de liens doit avoir résolu toutes les références.
- Si ce n'est pas le cas, il signale une erreur. Le programme ne peut pas s'exécuter.
- Si l'éditeur de liens n'effectue pas le chargement en même temps, alors il faut passer par le chargeur pour obtenir un binaire exécutable.

L'éditeur de liens

Exemple : la commande gcc

permet d'appeler à la fois le compilateur, l'éditeur de liens et le chargeur.
Programme composé de deux parties `prog1.c` et `prog2.c` :

- `gcc prog1 prog2`
- `gcc -c prog1.c prog2.c` appelle le préprocesseur, effectue la compilation et l'assemblage, mais pas l'édition de liens. Seuls les fichiers objets (*.o) sont générés.
- `gcc -o prog prog.o prog1.o` fixe le nom du fichier objet généré lors de la compilation.

Utilisation de bibliothèques :

- `gcc -o prog -llib prog.o` indique au compilateur qu'il doit utiliser la bibliothèque `lib`

L'édition de liens

Ce qu'on a décrit, c'est une édition de liens statique.

Édition de liens dynamique

- Le remplacement des adresses relatives par des adresses absolues a lieu au moment de l'exécution.
- Réalisée au niveau du matériel.
- Charge le registre de base du programme dans un registre spécial, utilise un mécanisme d'adressage fourni par le matériel, qui ajoute sa valeur à toute adresse référencée.
- permet de déplacer le programme d'une zone à une autre en cours d'exécution.

L'édition de liens

Édition de liens statique/dynamique

Statique :

- programme auto-suffisant
- taille

Dynamique :

- taille
- surcoût : édition de liens effectuée lors de chaque exécution