

Production et exécution de programmes

Catherine Faron Zucker, UNSA, EPU SI1
faron@polytech.unice.fr

1

Production de programmes

- Langage de programmation de haut niveau versus langage machine
 - Modèle de calcul abstrait, plus proche des applications
 - Structuration des données de la machine sous-jacente:
Types de données et opérations
 - définis dans le langage (int, double, ...)
 - définis par l'utilisateur (struct, class, ...)
 - Structuration du flot de contrôle (fonctions)
 - Dissociation de la spécification et de l'implémentation

Production de programmes

- Langage de programmation de haut niveau versus langage machine
 - Sécurité de programmation
 - Vérifications statiques
 - Avant l'exécution: lors du processus de traduction
exemple: analyse de type
 - Vérifications dynamiques
 - Pendant l'exécution
exemple: dépassement de la borne d'un indice

Production de programmes

- Traduction en langage machine d'un code source écrit dans un langage de haut niveau
 - Compilation
 - Le code source est traduit une fois pour toutes
 - Le résultat de cette traduction est un fichier binaire exécutable
 - Ce fichier est chargé en mémoire à chaque exécution
 - Interprétation
 - Le code source est traduit à la volée, instruction par instruction
 - Cette traduction a lieu à chaque exécution

Compilation versus interprétation

■ Compilation

- ⊙ Vérifications statiques et dynamiques
- ⊙ Optimisations statiques
- ⊗ Programme figé
- ⊗ Cycle de développement lourd

■ Interprétation

- ⊙ Grande puissance d'expression
- ⊙ Programme évolutif
- ⊙ Mise au point interactive
- ⊗ Vérifications dynamiques seulement
- ⊗ Pas d'optimisation

Compilation *et* interprétation

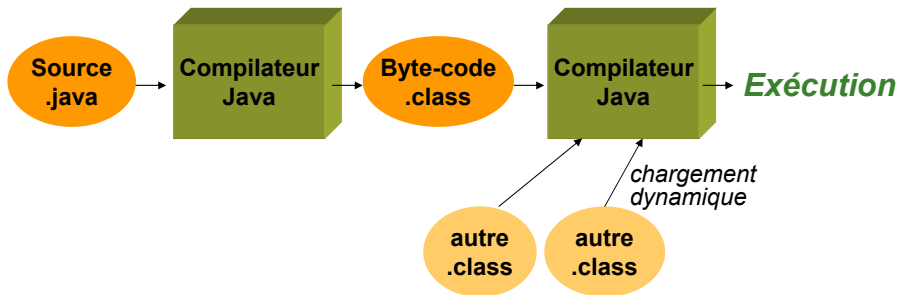
■ Langage de haut niveau et matériel

- Un source `.c` est traduit (compilé) en langage machine
- Le processeur interprète ce langage machine
- Les exécutables sont dépendants du matériel

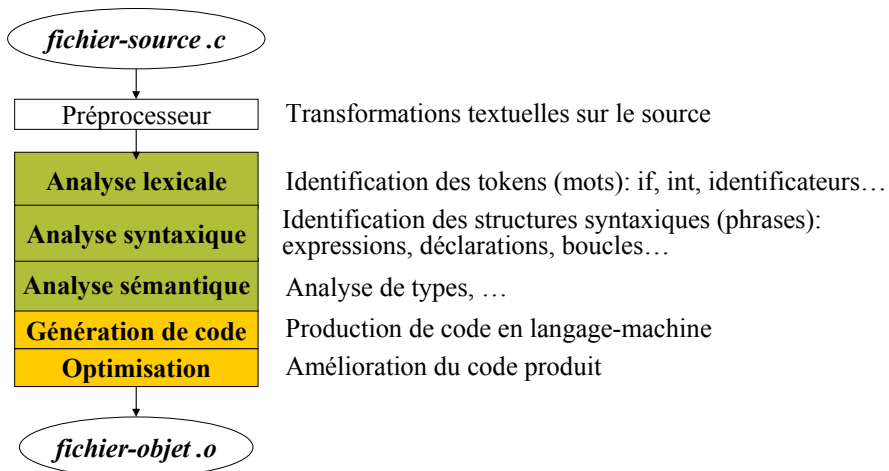
■ Langage de haut niveau et machine virtuelle

- Un source `.java` est compilé en langage machine *virtuel* (byte-code), le résultat est un fichier `.class`
- Un fichier `.class` est interprété par un programme appelé machine virtuelle (JVM)
- Le byte-code est indépendant du matériel (donc portable)

Compilation *et* interprétation



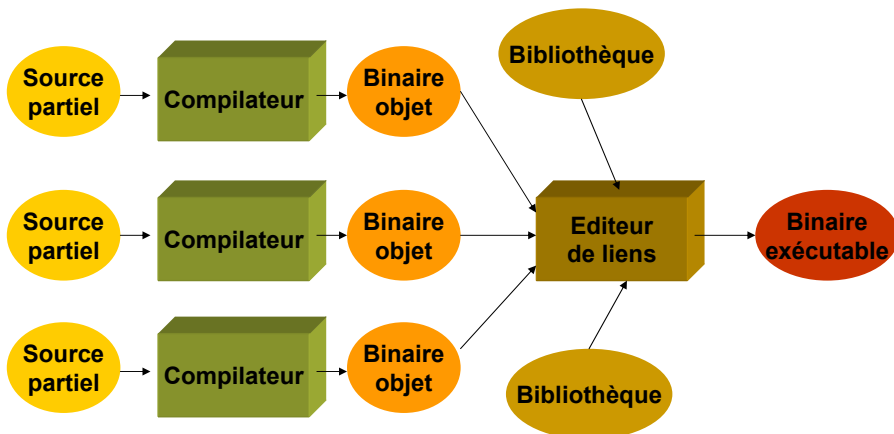
Processus de compilation



Compilation séparée

- Fichier source unique
 - Difficile à maintenir, inefficace
 - Besoin de bibliothèques précompilées
- Compilation séparée
 - Le source du programme est découpé en unités de compilation
 - En C, un fichier `.c` et tous les fichiers `.h` qu'il inclut
 - En Java, un fichier `.java`
 - Chaque unité est compilée séparément
 - Un éditeur de lien « colle les morceaux » et produit l'exécutable

Compilation séparée et édition de liens

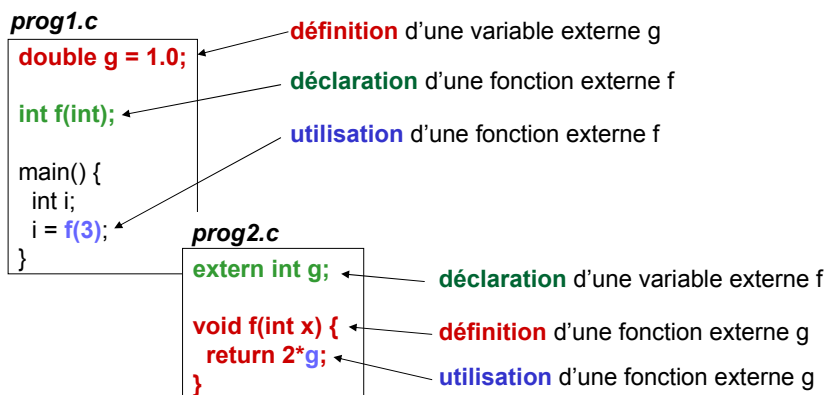


Edition de liens

- Lier ensemble les différentes parties d'un programme
 - Fichiers-objets produits par la compilation séparée
 - Bibliothèques (fichiers-objets précompilés)
- Résolution de références
 - Utilisation d'un objet (constante, variable, fonction) dans une unité de compilation alors qu'il est défini dans une autre unité
 - Ne concerne que les fonctions et les variables ou constantes statiques externes

Edition de liens

■ Résolution de références (2)

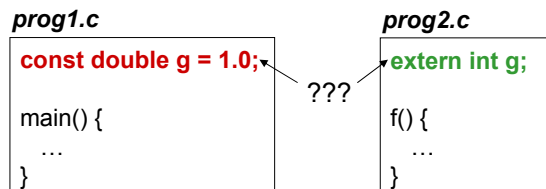


Edition de liens

- Résolution de références (3)
 - Chaque fichier-objet contient une table des symboles externes afin de faciliter le travail de l'éditeur de liens
 - Commandes (unix/linux) de lecture des fichiers-objets
 - **nm** : liste des symboles
 - **objdump**, **readelf**: lisent le contenu

Edition de liens et typage

- Le typage est réalisé par le compilateur
- En cas de compilation séparée
 - Chaque fichier-source doit contenir toute l'information nécessaire aux vérifications statiques
 - Les fichiers-objets ne contiennent plus d'information liée au typage

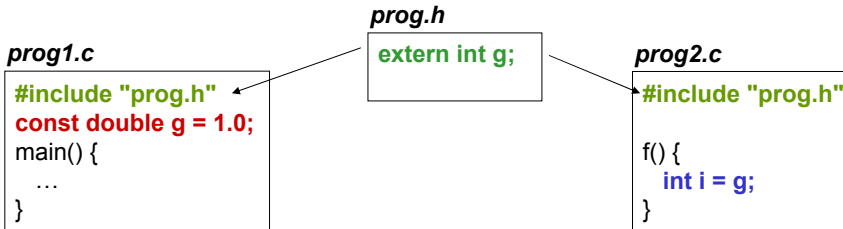


Edition de liens et typage

■ La bonne pratique :

Donner les moyens au compilateur de vérifier le typage

- Déclarer les variables externes dans des fichiers en-tête .h
- Inclure ces fichiers .h dans les fichiers sources .c qui utilisent ou définissent ces variables



Bibliothèques, « Libraries »

- Fichier unique contenant un ensemble de fichiers-objets précompilés
- L'éditeur de liens traite les bibliothèques de manière spéciale
 - Seul le code nécessaire au programme (les définitions d'objets utilisés par le programme) ira dans le binaire exécutable
- Commandes unix/linux
 - **ar** : crée une bibliothèque par regroupement de .o
 - **ld** : crée une bibliothèque (partagée)
 - **ranlib** : dote la bibliothèque d'un index (aide pour ld)
 - **ar**, **nm**, **readelf** : lisent le contenu d'une bibliothèque

Bibliothèques partagées

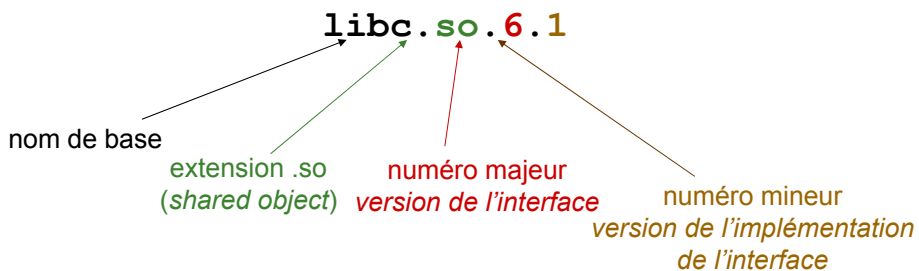
■ Motivations

- ❑ Eviter la duplication du code des bibliothèques courantes (e.g., la bibliothèque C standard libc)
 - Dans le fichier binaire exécutable
 - En mémoire
- ❑ Permettre de modifier l'implémentation d'une bibliothèque (mais pas son interface !)
 - Sans refaire l'édition de liens des applications
- ❑ Permettre le chargement à la demande de bibliothèques

Bibliothèques partagées

■ Nommage et version

- ❑ Sous Windows
 - Nom avec extension `.dll` (*dynamically linked library*)
- ❑ Sous Unix/Linux



Edition de liens statique

■ Création de la bibliothèque

```
gcc options -c libobj1.c
gcc options -c libobj2.c
...
ar c libobj.a libobj1.o libobj2.o ...
ranlib libobj.a
```

■ Utilisation de la bibliothèque

```
gcc options -o myprog prog1.o prog2.o libobj.a
OU
gcc options -o myprog prog1.o prog2.o -L dir -lobj
```

Edition de liens dynamique

■ Création de la bibliothèque

```
gcc options -fPIC -c libobj1.c
gcc options -fPIC -c libobj2.c
...
gcc -shared -Wl,soname,libobj.so.3 \
    -o libobj.so.3.2 libobj1.o libobj2.o ...
```

■ Utilisation de la bibliothèque

```
gcc options -o myprog prog1.o prog2.o libobj.so
OU
gcc options -o myprog prog1.o prog2.o -L dir -lobj
OU
```

Chargement dynamique

Chargement dynamique de bibliothèques

- Chargement/déchargement à la demande d'une bibliothèque sans édition de liens préalable avec elle
- Sous **unix/linux**, utilisation d'une bibliothèque de chargement dynamique, **libdl.so** (fichier en-tête **<dlfcn.h>**)
 - dlopen()** ouverture d'une bibliothèque partagée
 - dclose()** fermeture de la bibliothèque
 - dlsym()** recherche d'un symbole dans la bibliothèque
 - dLError()** traitement des erreurs
- Mécanisme similaire au chargement dynamique de classes en Java

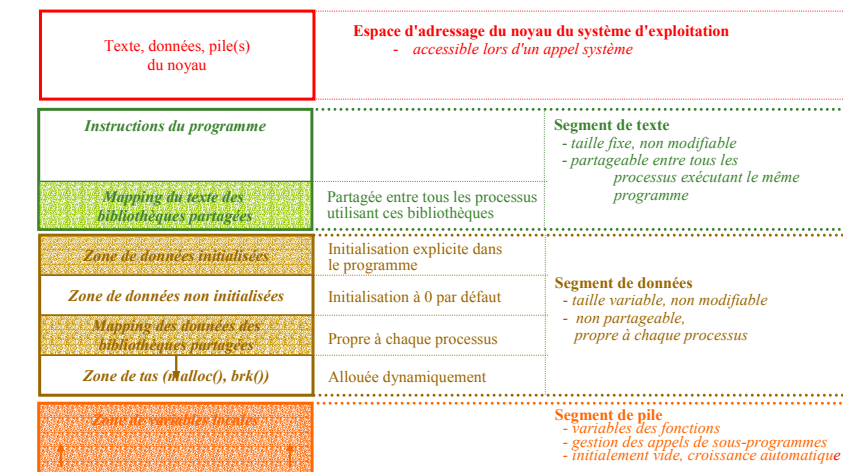
Manipulation des bibliothèques partagées

- Lecture du contenu des bibliothèques partagées
 - **readelf**
- Bibliothèques partagées utilisées par un exécutable ou une autre bibliothèque
 - **ldd**
- Recherche des bibliothèques partagées à l'exécution
 - Chargeur dynamique : **ld.so**
 - Variable d'environnement **LD_LIBRARY_PATH**
 - **ldconfig**

ELF: Executable and Linking Format

- Remplace l'ancien format `a.out`
- Format de fichier unique
 - pour exécutables, objets, et bibliothèques partagées
 - Fichier exécutable (*executable file*) : information nécessaire à l'OS pour créer l'image mémoire d'un processus
 - Fichier relogeable (*relocatable file*) : destiné à subir une édition de liens avec d'autres fichiers objets pour créer un exécutable ou une bibliothèque partagée
 - Fichier objet partagé (*shared object file*) : information pour l'édition de liens statique ou dynamique
- Lecture des informations d'un fichier ELF
 - `readelf`, `libelf`

Espace d'adressage d'un processus



Runtime

- Run time (versus compile time)
 - Période pendant laquelle un programme s'exécute
- Runtime et modèle d'exécution
 - Code nécessaire à l'exécution d'un programme
 - Code nécessaire à l'implémentation de la sémantique d'exécution du langage dans lequel un programme est écrit

Runtime

- Code généré par le compilateur
 - Structures de contrôle, appels de fonction
 - Evaluation des expressions
- Bibliothèques runtime
 - Déclenchement des appels-système
 - Gestion mémoire (e.g., `malloc()`, `new()`)
- Machines virtuelles
 - Interprète de bytecode
 - Classloader

Bibliothèques runtime

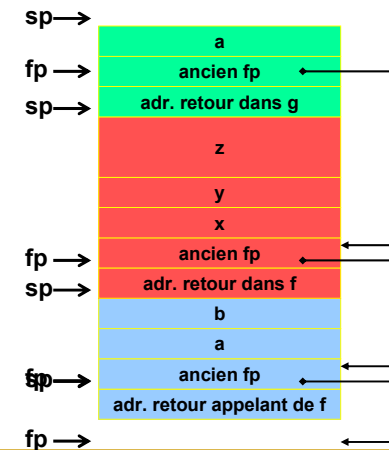
- Gestion mémoire dynamique
 - Segment de données
 - Allocation de mémoire dans la zone de tas (*heap*)
 - Mémoire partagée entre processus
 - Données des bibliothèques dynamiques
 - Segment de texte
 - Code des bibliothèques dynamiques

Bibliothèques runtime

- Gestion de la pile
 - Gestion de la chaîne dynamique d'appels de fonctions
 - Gestion des variables locales des fonctions
 - Gestion des exceptions

Bibliothèques runtime

■ Gestion de la pile



```
void f() {  
    int a, b;  
    g(a);  
}  
void g(int x, int y) {  
    double z;  
    h(z);  
}  
void h(double a) {  
    ...  
}
```

Machine virtuelle

Principe :

- Inventer une architecture de processeur adaptée à l'interprétation d'un langage de haut niveau
 - Registres
 - Jeu d'instructions
 - Structures de données manipulées ...
- Fournir une réalisation *logicielle* de cette machine (émulateur)
- Compiler le langage de haut niveau dans le jeu d'instructions de la machine

Machine virtuelle

Avantages et inconvénients

😊 Portabilité

Seule la machine virtuelle doit être réécrite pour les différentes architectures

😊 Avantages de l'interprétation et de la compilation réunis

Vérifications statiques et dynamiques

😞 Performance

Interprétation logicielle

Double traduction

😞 Maintenance

Un changement du langage implique la mise à jour du compilateur et de la machine virtuelle