Architecture et Systèmes

Stefan Schwoon

Cours L3, 2025/2026, ENS Paris-Saclay

Mémoire et exécutables

Contenu:

- Organisation mémoire dans plusieurs architectures (C64, IBM PC, Unix)
- Format des exécutables
- Liens dynamiques

Premier exemple : Commodore C64

Ordinateur populaire dans les années 1980

Source : Wikimedia Commons, auteur : Evan-Amos

Mémoire : 64 ko

Agit avec des périphéries bien définis, système d'exploitation et interpréteur BASIC intégré dans le ROM.

Architecture et binaires

CPU : registres de 8 bit, pas de mode privilégié

organisation mémoire fixe, avec des zones reservés pour l'interpreteur BASIC et le programme actuelle

`LOAD "file",8` charge un fichier BASIC depuis la datasette

`LOAD "file",8,1` charge un binaire dans une adresse donnée par `file`

Second exemple : IBM PC

Ordinateur professionnel conçu par IBM en 1981

Source : Wikimedia Commons, auteur : Boffy b

Mémoire : 1 Mo adressable, dont 640 ko de RAM

Système modulaire, extensible avec des cartes périphériques developpé par des tiers partis.

Segmentation

Registres de 16 bits:

- Registres données : AX, BX, CX, DX
- Registres segments : CS, DS, SS, ES (code, data, stack, extra)
- Registres index : SP, BP, SI, DI
- Compteur : IP

Adresse interprétées avec une paire segment+index/donnée, p.ex. CS:IP

$$\mathrm{CS} * 16 + \mathrm{IP}$$

Organisation mémoire

Premier ko: 256 vecteurs d'interruption (paires segment/index), utilisées lors des instructions INT xx (ou interruption matériel)

Partie au dela de 640k: ROM / carte graphique

Reste: géré par le système d'exploitation (typiquement MS-DOS)

- charge d'abord le système d'exploitation en mémoire + pilotes de périphériques
- possibilités de TSR (terminate-and-stay-resident), après avoir détourné des interruptions

Note : pas de concurrence (système single-tâche), ni mode privilégié
(donc pas de protection mémoire)

Exécutables COM

Format binaire simple (p.ex. `command.com`)

Code+données limités à 64 ko, donc adressable avec un même registre segment

Fichier sur disque = image mémoire, les instructions manipulent des données dans un même segment

En lançant l'exécutable, le système charge l'image à une adresse libre, puis lance le code avec les registres de segment bien initialisés

Exécutables EXE / MZ

Format binaire relocalisable

L'image sur disque définit plusieurs zones de mémoire et leur taille.

Chaque zone est chargé dans un segment différent.

Le code stoqué sur disque prétend que le code est chargé au segment 0.

Le système stocke l'image dans un segment libre, puis consulte une *liste de relocalisations* dans le fichier EXE. Chaque adresse dans la liste est ensuite augmenté par l'adresse du segment choisi par le système.

Détails techniques : `https://wiki.osdev.org/MZ`

Attention : les fichiers dans les versions de Windows actuelles s'appellent toujours EXE mais fonctionnent différemment.

Exécutables dans les systèmes modernes

Besoins : Multi-tâche, protection de mémoire $\rightarrow$ mémoire virtuelle, bibliothèques partagées, adaptations aux architectures différentes (32/64 bits, petit/grand boutisme)

Unixoid : format ELF (Executable and Linking Format)

MacOS : format Mach-O

Windows : EXE / PE (portable executable)

Focus sur ELF (utilisé pour exécutables / bibliothèques / objects / cores etc.)

Format ELF : éléments clés

Un fichier ELF est composé d'une entête de 64 octets, une liste d'*entêtes de programmes* (program header table) et une liste d'*entêtes se sections* (section header table), puis les données associés aux programmes et sections.

L'entête commence par des octets `7f 45 4c 46`, puis une indication si le format est de 32/64 bits, le boutisme, le système et processeur visé. Ensuite les adresses (dans le fichier) des deux listes d'entêtes, leur tailles et nombres d'entrées.

Détails techniques :

`https://wiki.osdev.org/ELF`

ELF cheat sheet

Décryptage d'un fichier : commande `readelf`

Format ELF : entête de programme

Entête de programme : identifie une partie du fichier exécutable qui doit être chargée en mémoire.

Contient le déplacement dans le fichier, la taille, l'adresse virtuelle et les droits d'accès (lecture, écriture, exécutable).

Entête de section : identifie les fonctions de certains données

Une section peut être une partie des données chargées en mémoire, ou encore une partie du fichier exécutable.

Elle porte un nom qui commence typiquement avec un point (.)

Types de sections

Quelques sections importantes :

- `.shstrtab` : contient les noms des sections (indiqué dans l'entête ELF)
- `.init`, `.fini` : code à exécuter avant/après la partie principale du code
- `.text` : partie principale du code (adresse indiqué dans l'entête ELF)
- `.data` : données intialisées
- `.symtab`, `.strtab` : tableaux de symboles (pour compilation distribuée)
- `.dynsyn`, `.dyntab`, `.rela.*` : édition de liens dynamiques
- `.debug_*` : noms des variables, relation assembleur/lignes C etc